

# Основы глубинного обучения

Лекция 3

Свёрточные сети

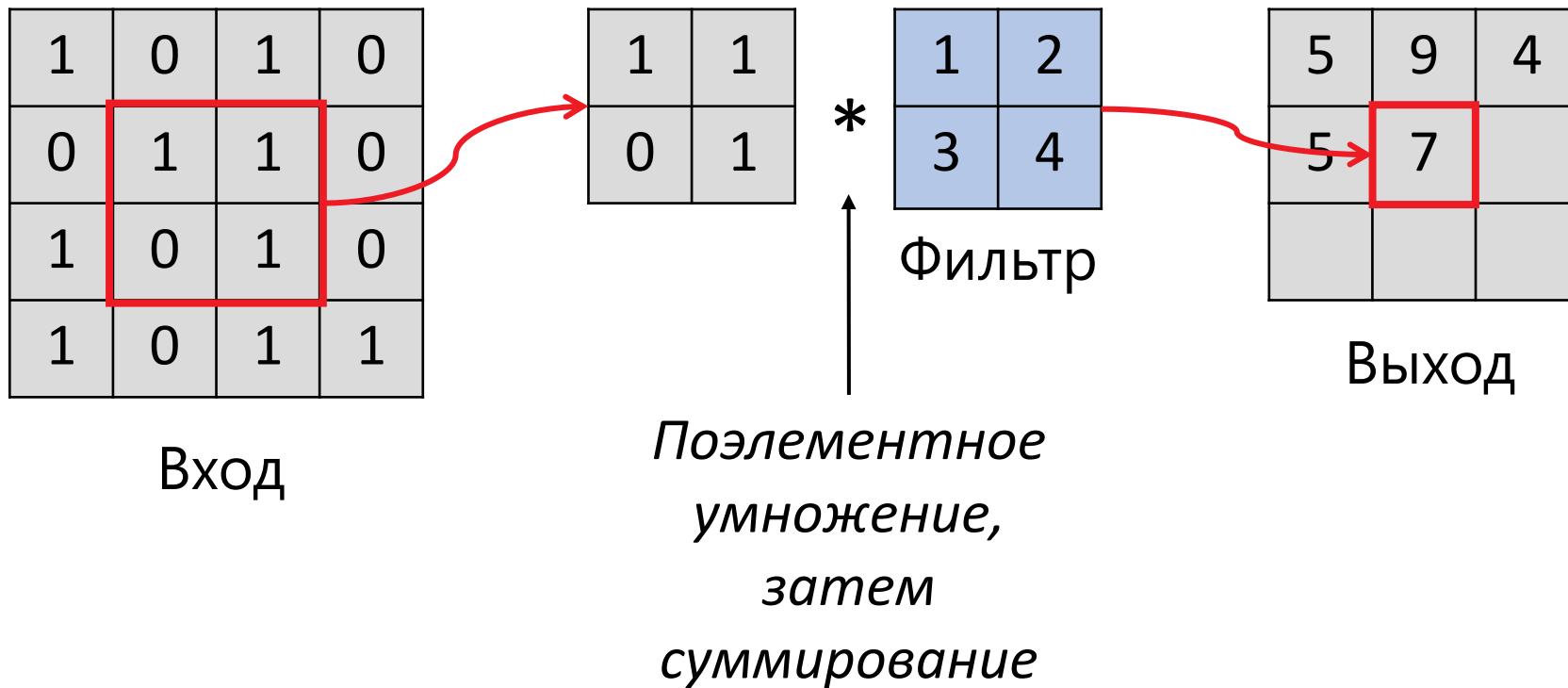
Евгений Соколов

[esokolov@hse.ru](mailto:esokolov@hse.ru)

НИУ ВШЭ, 2025

# Свёртка

Поле восприятия  
(receptive field)



# Свёртка инвариантна к сдвигам

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Вход

\*

1	0
0	1

=

0	0	0
0	1	0
0	0	2

Фильтр

Выход

Max = 2

1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

Вход

\*

1	0
0	1

=

2	0	0
0	1	0
0	0	0

Фильтр

Выход

Не меняется

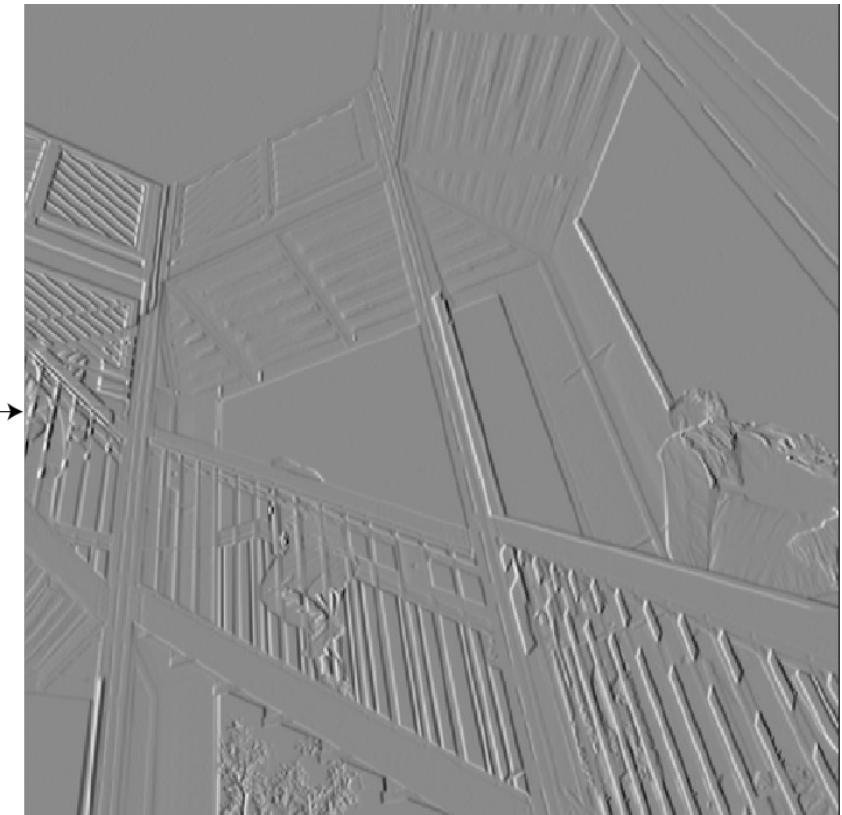
Max = 2

# Свёртки в компьютерном зрении



$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel



# Свёртка

$$\text{Im}^{out}(x, y) = \sum_{i=-d}^d \sum_{j=-d}^d (K(i, j) \text{ Im}^{in}(x + i, y + j) + b)$$

# Свёртка

$$\text{Im}^{out}(x, y) = \sum_{i=-d}^d \sum_{j=-d}^d (K(i, j) \text{ Im}^{in}(x + i, y + j) + b)$$

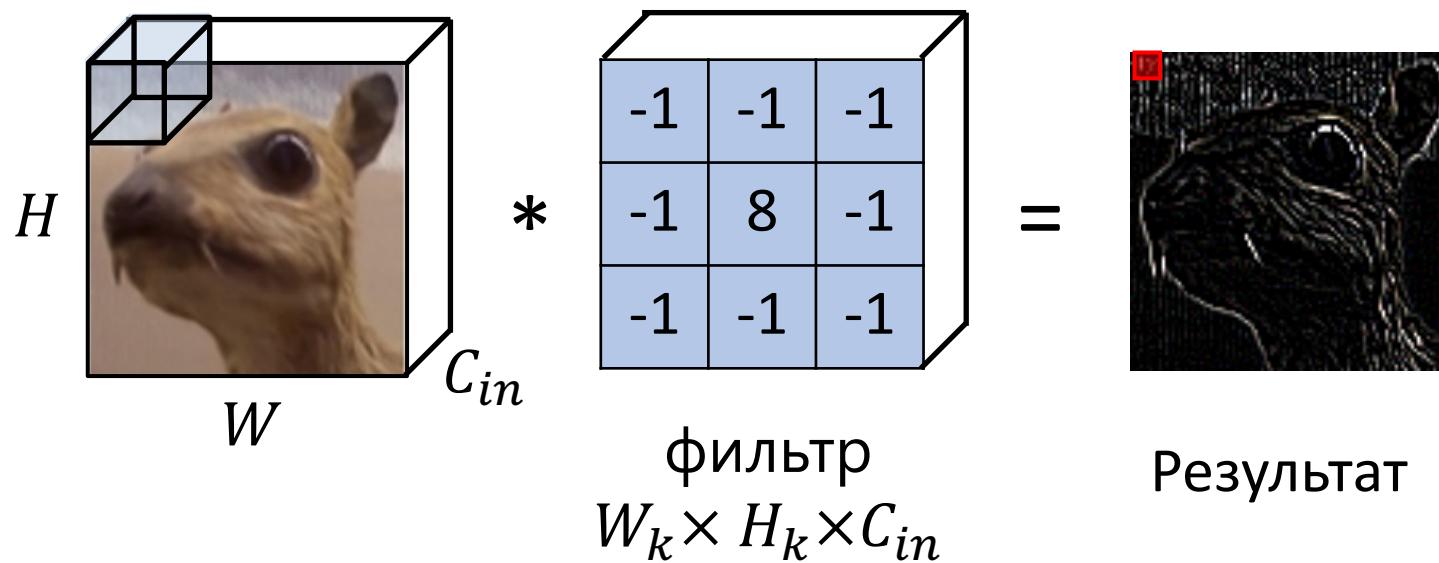
- Пиксель в результирующем изображении зависит только от небольшого участка исходного изображения (local connectivity)
- Веса одни и те же для всех пикселей результирующего изображения (shared weights)

# Свёртка

- Обычно исходное изображение цветное!
- Это означает, что в нём несколько каналов (R, G, B)
- Учтём в формуле:

$$\text{Im}^{out}(x, y) = \sum_{i=-d}^d \sum_{j=-d}^d \sum_{c=1}^C (K(i, j, c) \text{ Im}^{in}(x + i, y + j, c) + b)$$

# Свёртка



# Свёртка

- Одна свёртка выделяет конкретный паттерн на изображении
- Нам интересно искать много паттернов
- Сделаем результат трёхмерным:

$$\text{Im}^{out}(x, y, t) = \sum_{i=-d}^d \sum_{j=-d}^d \sum_{c=1}^C (K_t(i, j, c) \text{ Im}^{in}(x + i, y + j, c) + b_t)$$

# Число параметров

$$\text{Im}^{out}(x, y, t) = \sum_{i=-d}^d \sum_{j=-d}^d \sum_{c=1}^C (\textcolor{red}{K}_t(i, j, c) \text{ Im}^{in}(x + i, y + j, c) + \textcolor{red}{b}_t)$$

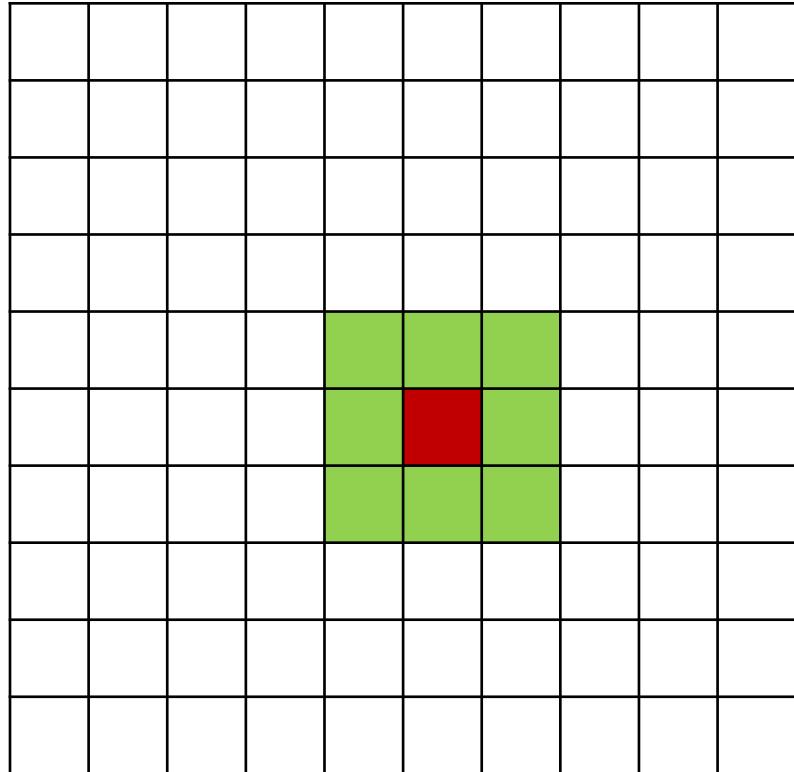
- Обучается только фильтр
- $((2d + 1)^2 * C + 1) * T$  параметров
- Как из этого сделать модель — обсудим позже

# Receptive field

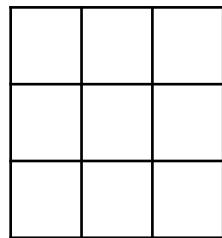
# Receptive field

- Возьмём пиксель в итоговом изображении (после свёрточных слоёв)
- От какой части входного изображения зависит значение в этом пикселе?

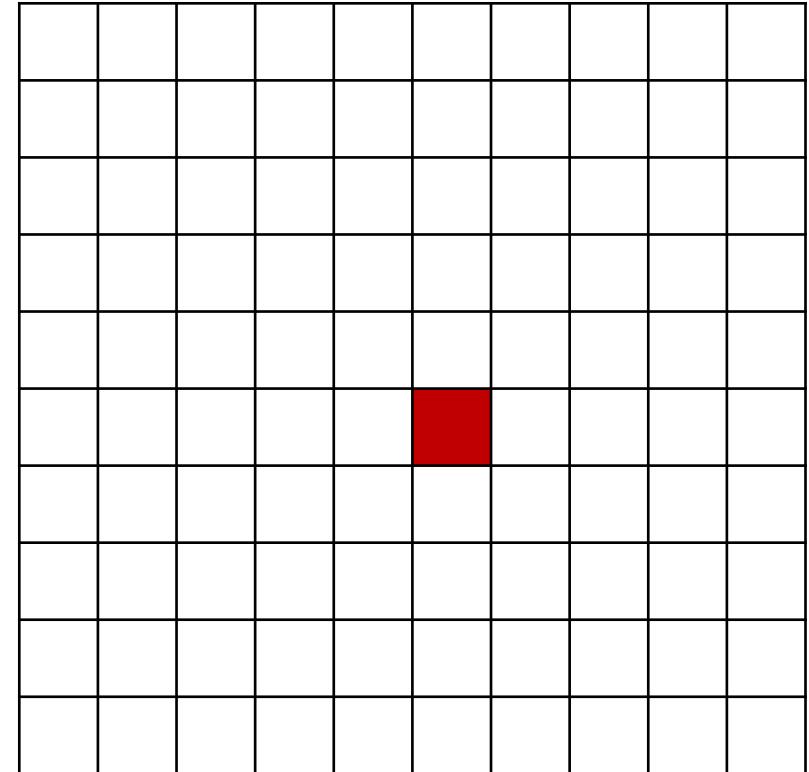
# Receptive field



\*

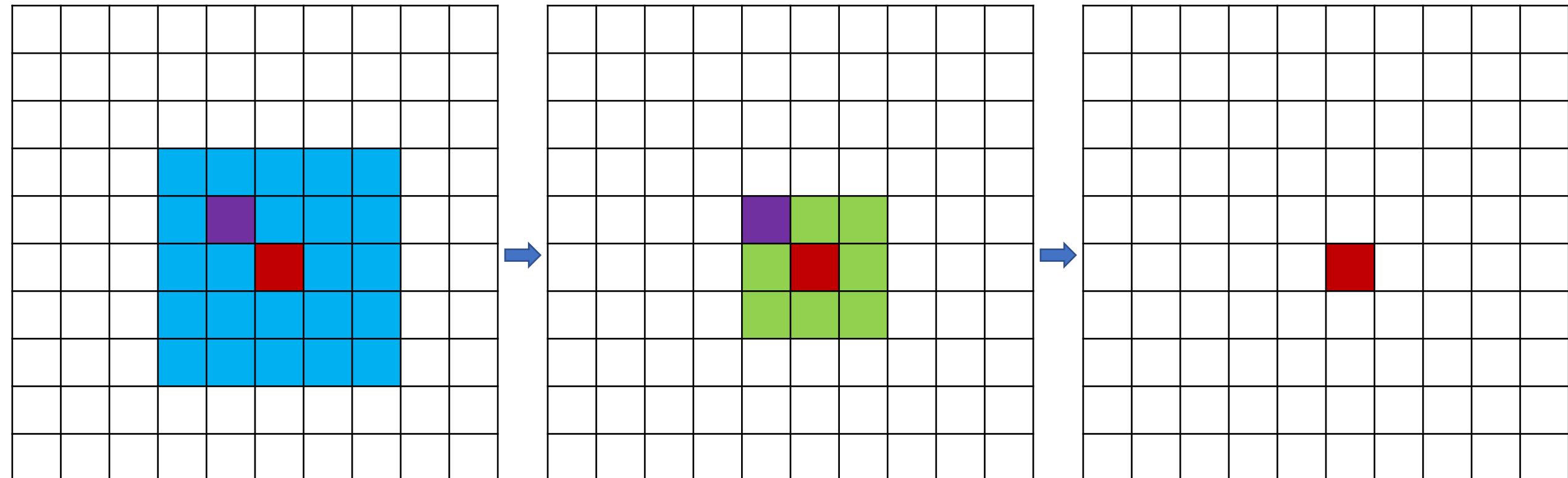


=



Поле восприятия:  $3 \times 3$

# Receptive field



Поле восприятия:  $5 \times 5$

# Receptive field

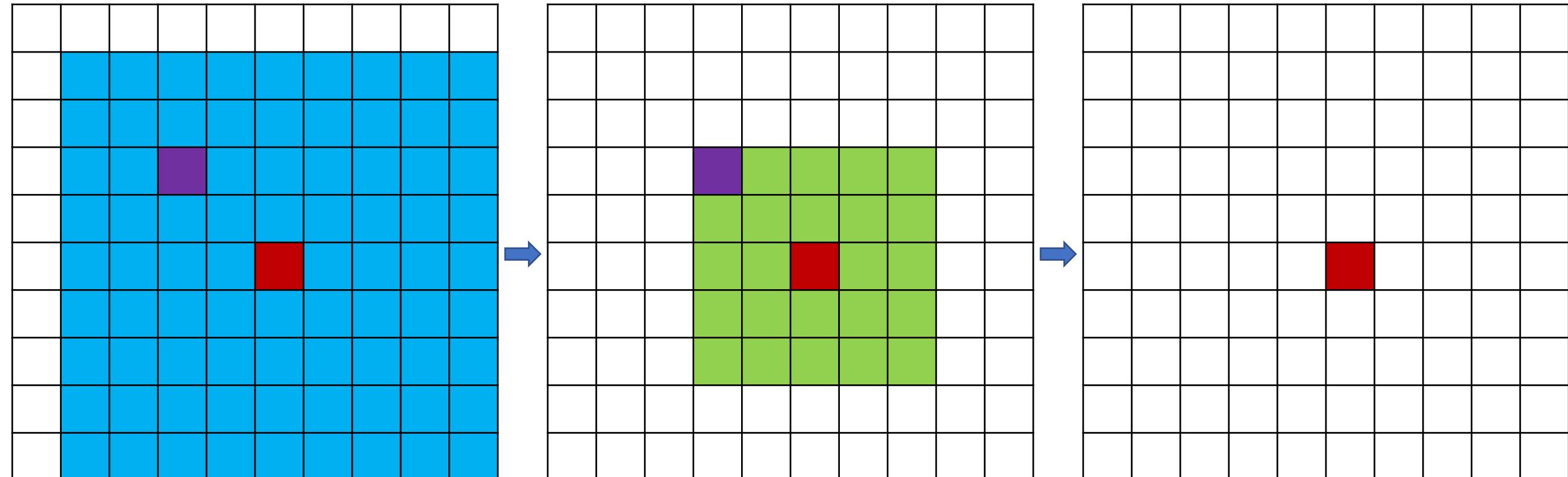
Поле восприятия для свёртки  $3 \times 3$ :

- После 1 свёрточного слоя:  $3 \times 3$
- После 2 свёрточных слоев:  $5 \times 5$
- После 3 свёрточных слоёв:  $7 \times 7$

# Receptive field

Поле восприятия для свёртки  $5 \times 5$ :

# Receptive field



Поле восприятия:  $5 \times 5$

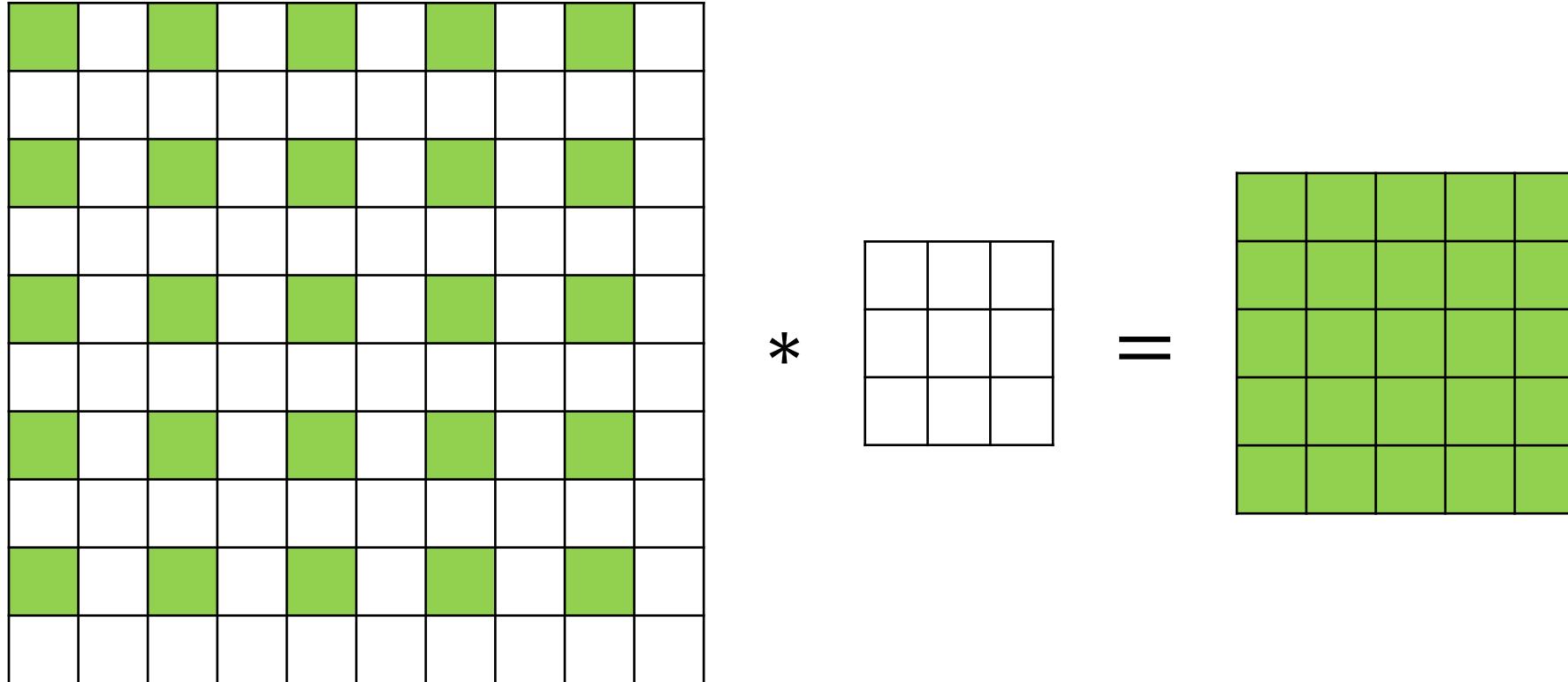
# Receptive field

Поле восприятия для свёртки  $5 \times 5$ :

- После 1 свёрточного слоя:  $5 \times 5$
- После 2 свёрточных слоев:  $9 \times 9$
- После 3 свёрточных слоёв:  $13 \times 13$

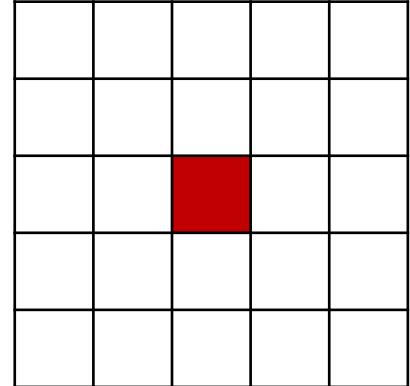
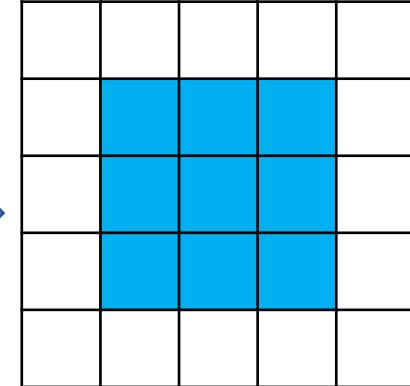
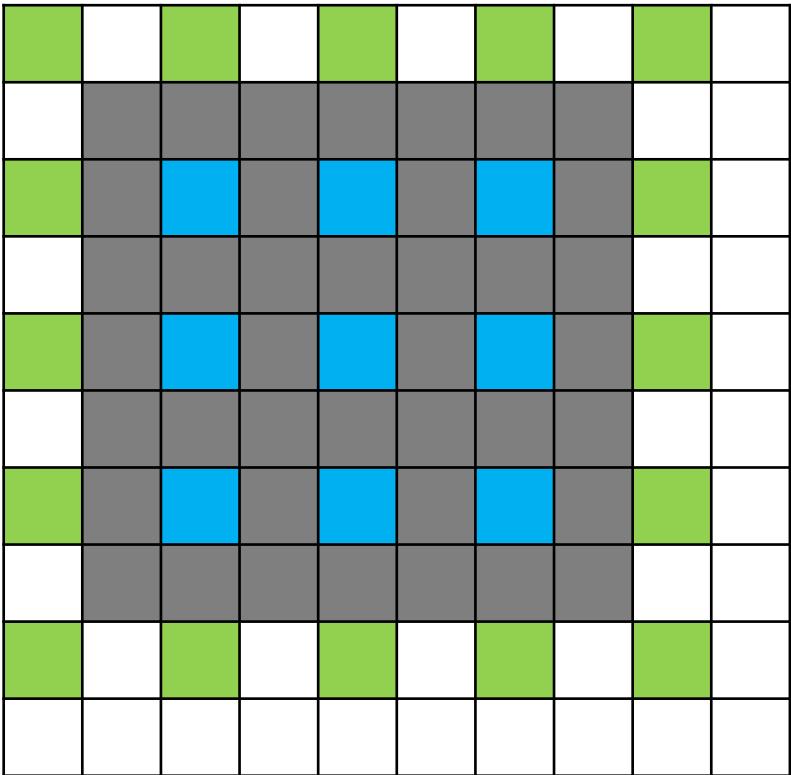
Нужно очень много слоёв, если изображение размера  $512 \times 512$

# Свёртки с пропусками (strides)



$$s = 2$$

# Свёртки с пропусками (strides)



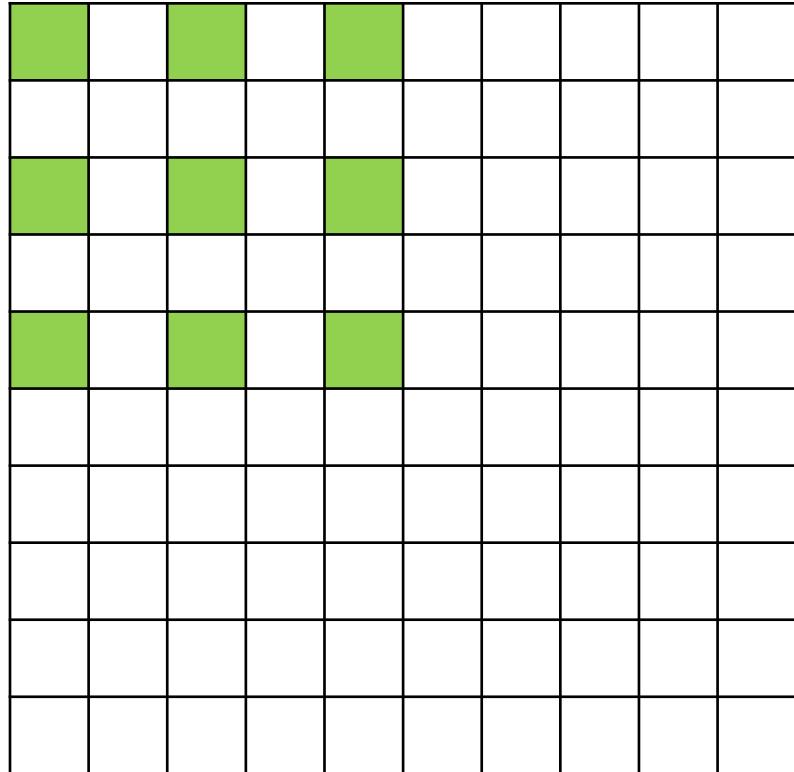
Поле восприятия:  $7 \times 7$

# Свёртки с пропусками (strides)

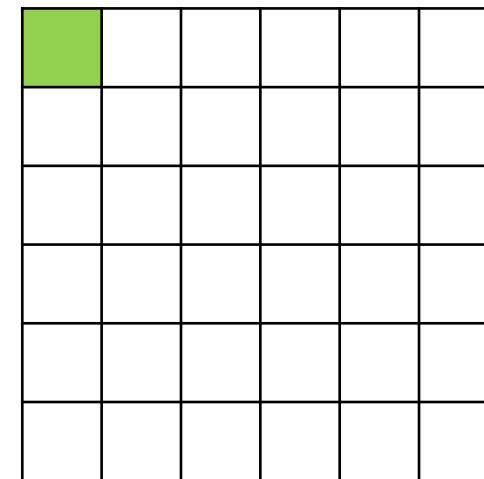
Подробности про подсчёт размера поля:

<https://distill.pub/2019/computing-receptive-fields/>

# Dilated convolutions («раздутые» свёртки)

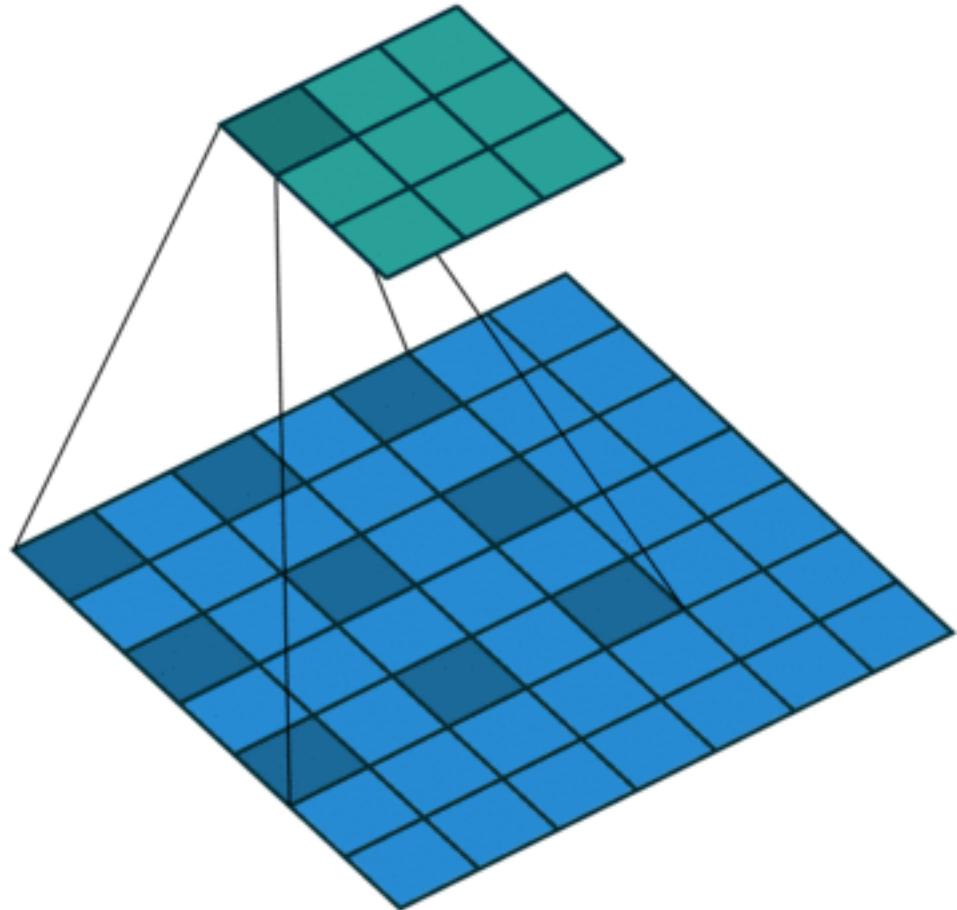


$$\begin{matrix} * & \begin{matrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{matrix} & = \end{matrix}$$

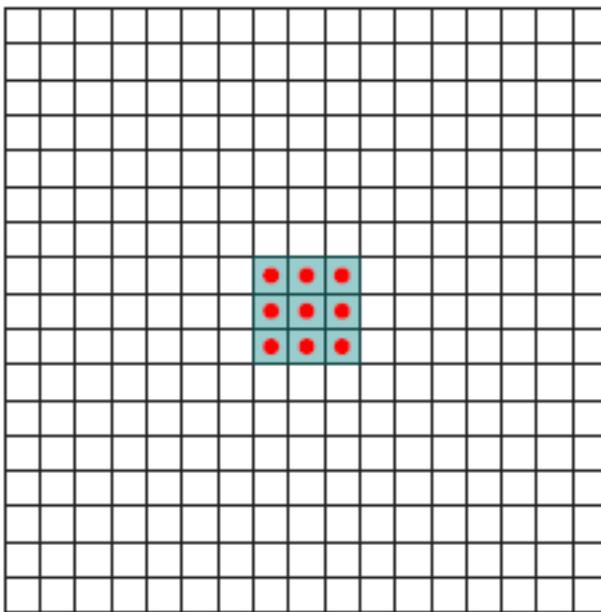


$$l = 2$$

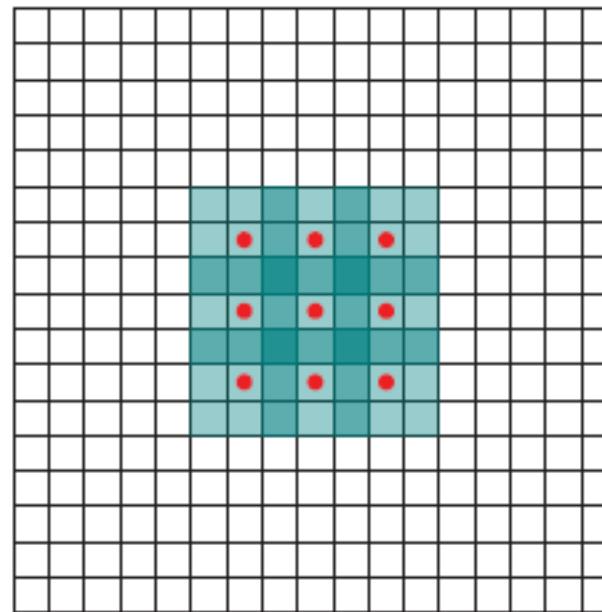
# Dilated convolutions («раздутые» свёртки)



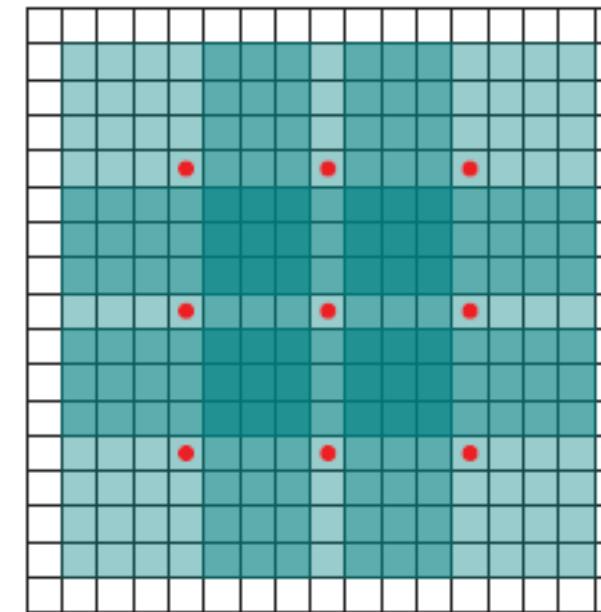
# Dilated convolutions («раздутые» свёртки)



$$l = 1$$



$$l = 2$$



$$l = 4$$

# Pooling

1	0	2	1	0	0
0	1	3	2	1	2



1	3	2

Max-pooling с фильтром 2x2

# Pooling

- Разбивает изображение на участки  $n \times m$  и считает некоторую статистику в каждом участке (обычно максимум)
- Существенно сокращает размер изображения (значит, увеличивает поле восприятия следующих слоёв)
- Не имеет параметров

## Зачем это всё?

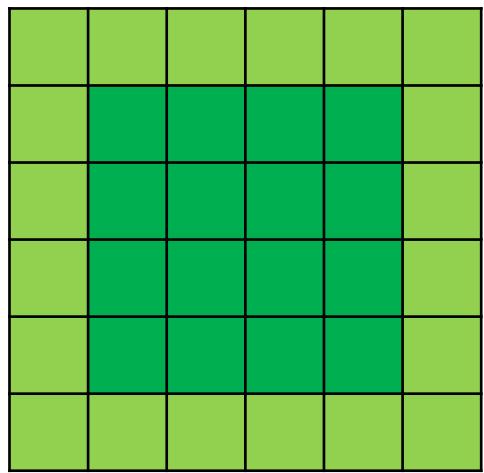
- Важно следить за тем, чтобы последние свёрточные слои имели размер поля восприятия, сравнимый со всей картинкой

# Padding

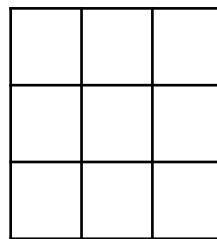
# Свёртки

- Если применять свёртку по формуле, то выходное изображение будет меньше входного

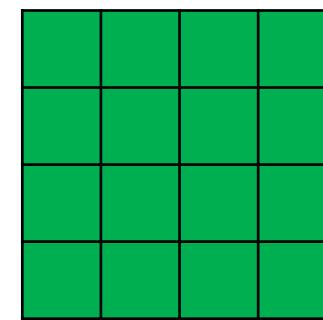
# Свёртки



\*



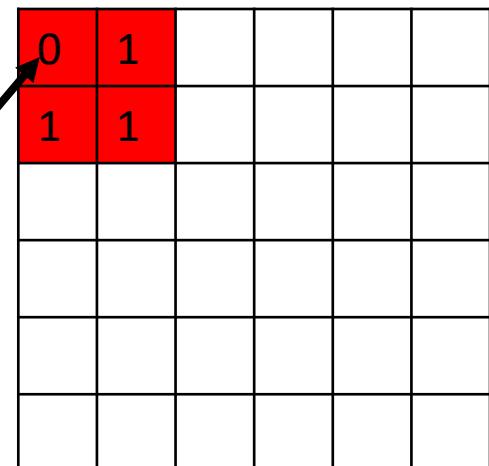
=



# Valid mode

- При честном подсчёте свёрток пиксели на краях не оказывают большого влияния на результат

Не увидим, что фильтр имеет хороший отклик при помещении центра в этот пиксель



A 5x5 grid of squares. The top-left square contains the value 1. An arrow points from the text "Не увидим, что фильтр имеет хороший отклик при помещении центра в этот пиксель" to this square. The square containing 1 is highlighted with a red background.

0	1			
1	1			

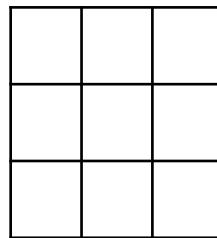
\*

0	0	1
0	0	1
1	1	1

# Zero padding

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

\*



=


# Zero padding

- Добавляем по границам нули так, чтобы посчитанная после этого свёртка в valid mode давала изображение такого же размера, как исходное
- Есть риск, что модель научится понимать, где на изображении края — можем потерять инвариантность

# Reflection padding

3	6	6	7	8					
8	7	1	2	3					
2	1	1	2	3	4	5	6		
7	6	6	7	8	9	8	7		
2	1	1	2	3					

\*


=


# Reflection padding

- Не получится легко находить края изображения
- Но теперь модель может начать находить зеркальные отражения и подбирать фильтры под них

# Replication padding

1	1	1	2	3				
1	1	1	2	3				
1	1	1	2	3	4	5	6	
6	6	6	7	8	9	8	7	
1	1	1	2	3				

\*


=


# Replication padding

- Пиксель на границе равен ближайшему пикслю из изображения
- Модель всё ещё может настроиться под паттерны, которые возникают из-за такого паддинга

# Резюме

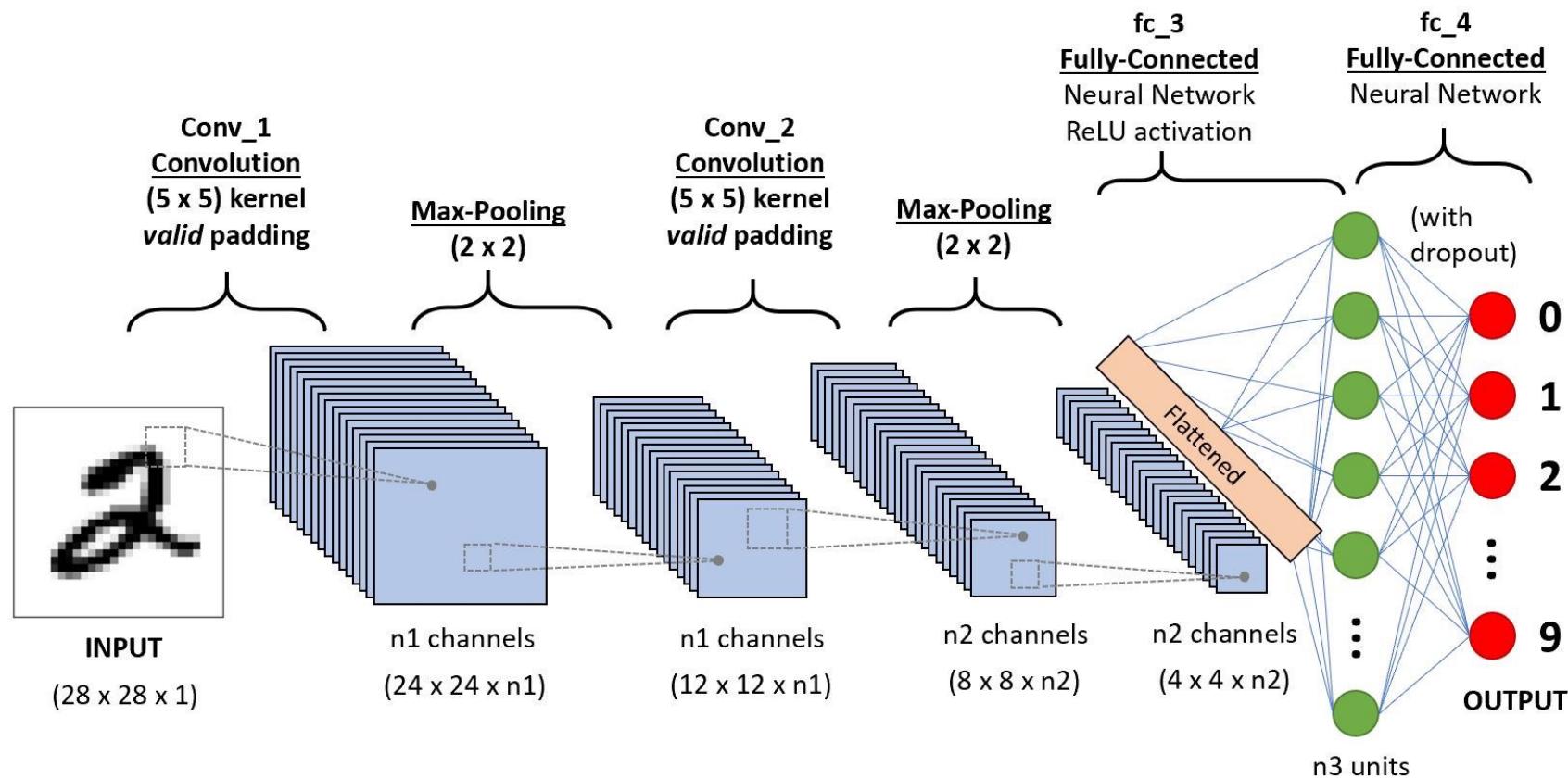
- Паддинг позволяет контролировать размер выходных изображений
- Паддинг позволяет учитывать даже объекты на краях
- Разные типа паддингов допускают разные способы переобучения под края

# Структура свёрточных сетей

# Свёрточный слой

$$\text{Im}^{out}(x, y, t) = \sum_{i=-d}^d \sum_{j=-d}^d \sum_{c=1}^C (K_t(i, j, c) \text{Im}^{in}(x + i, y + j, c) + \textcolor{red}{b}_t)$$

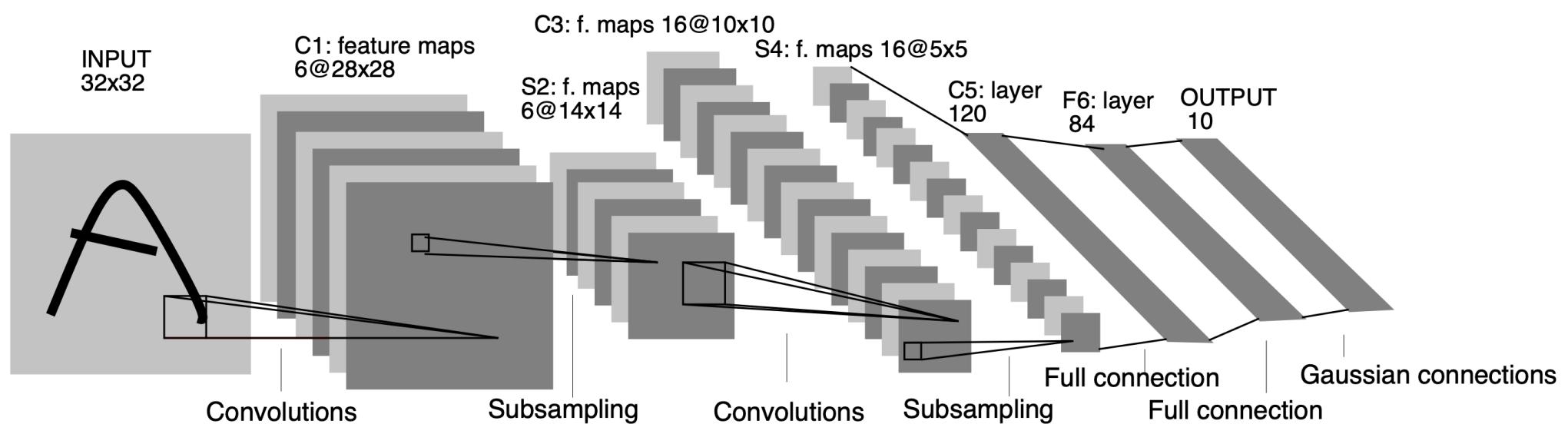
# Типичная архитектура



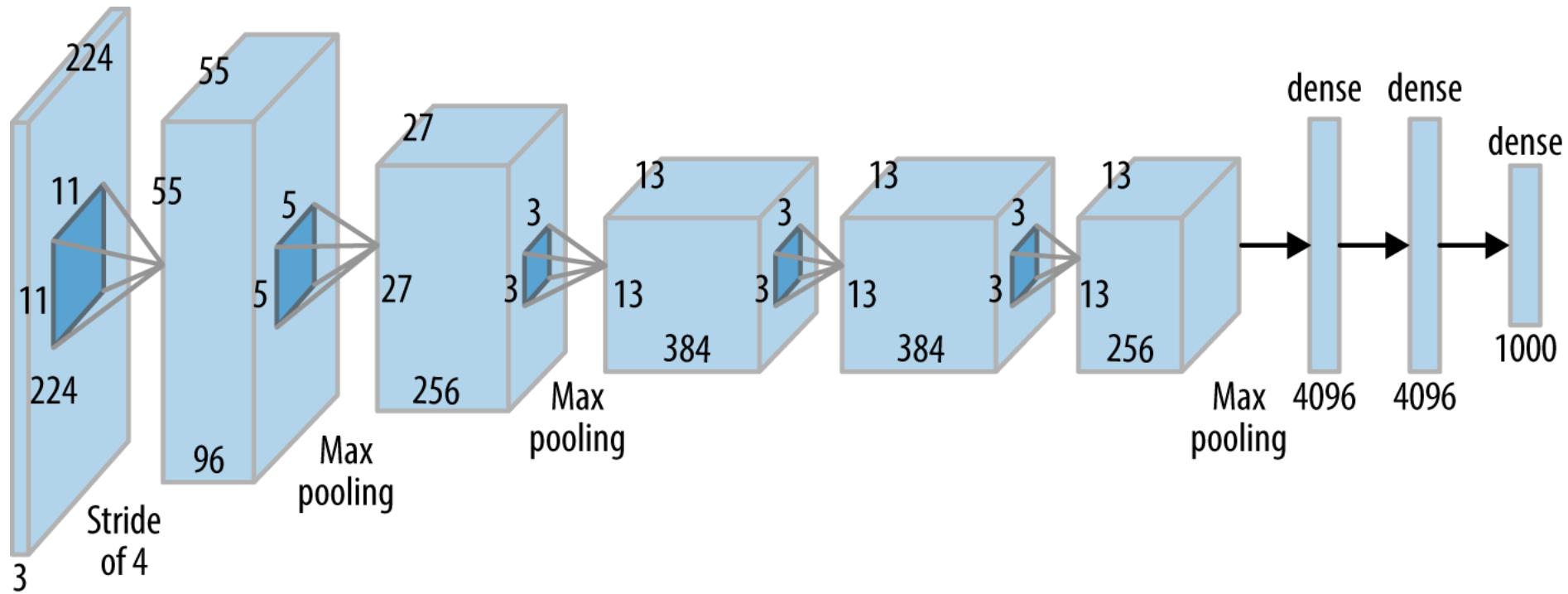
# Типичная архитектура

- Последовательное применение комбинаций вида «свёрточный слой -> нелинейность -> pooling» или «свёрточный слой -> нелинейность»
- Выпрямление (flattening) выхода очередного слоя
- Серия полносвязных слоёв

# LeNet



# AlexNet

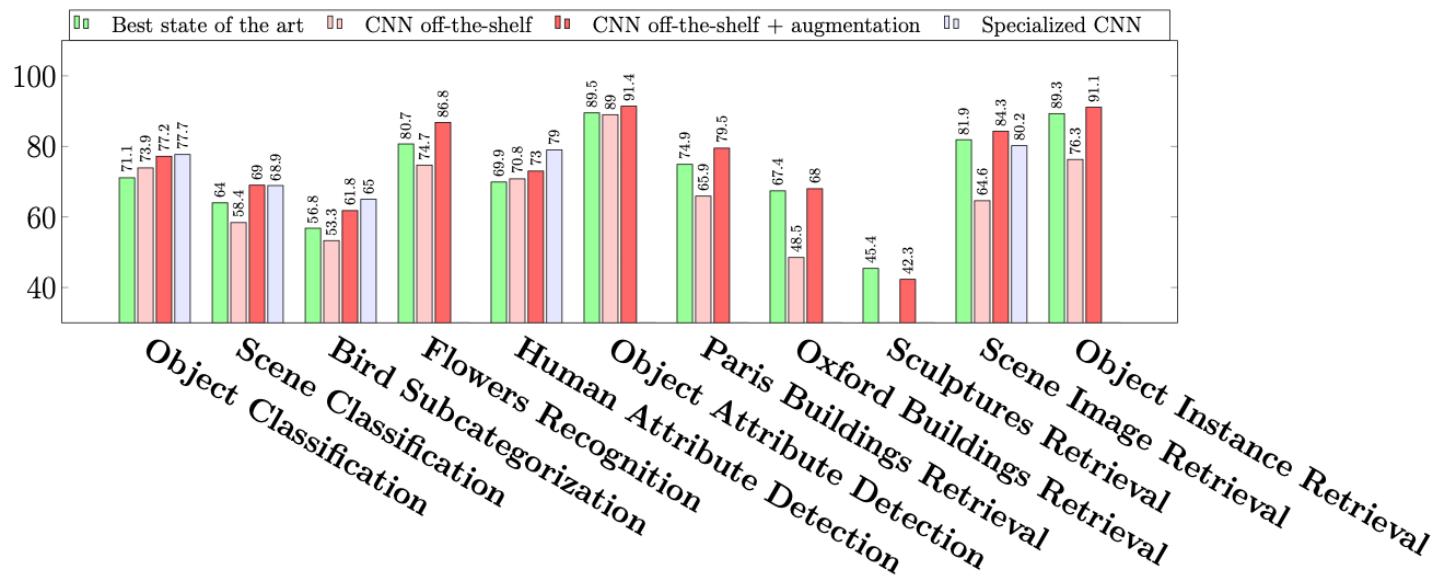
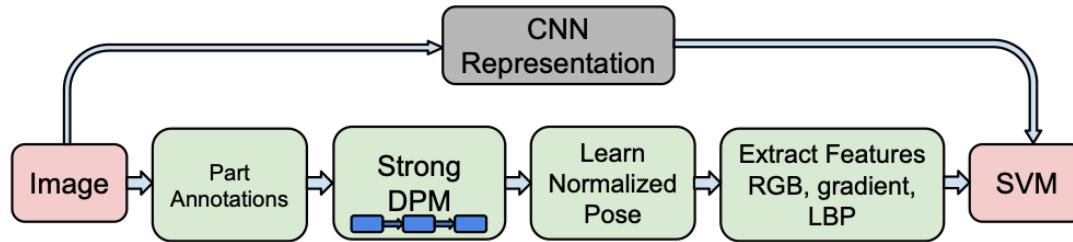


<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

# Представления с последних слоёв

- Важное наблюдение: выходы полносвязных слоёв являются хорошими признаковыми описаниями изображений
- Полезны во многих задачах
- Например, поиск похожих изображений

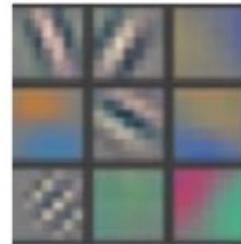
# Представления с последних слоёв



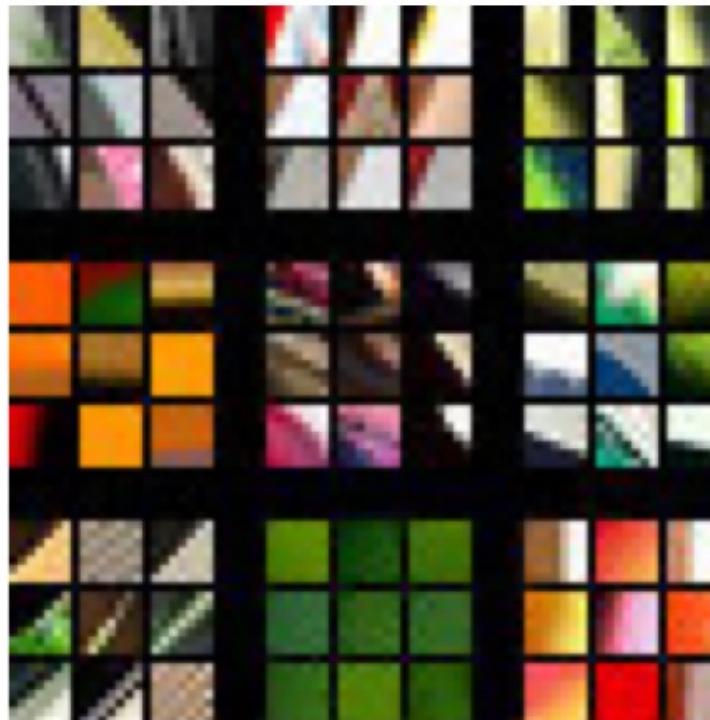
# Представления с последних слоёв

- Не интерпретируется (в отличие от классического компьютерного зрения)
- По смыслу — «индикаторы» наличия каких-то паттернов

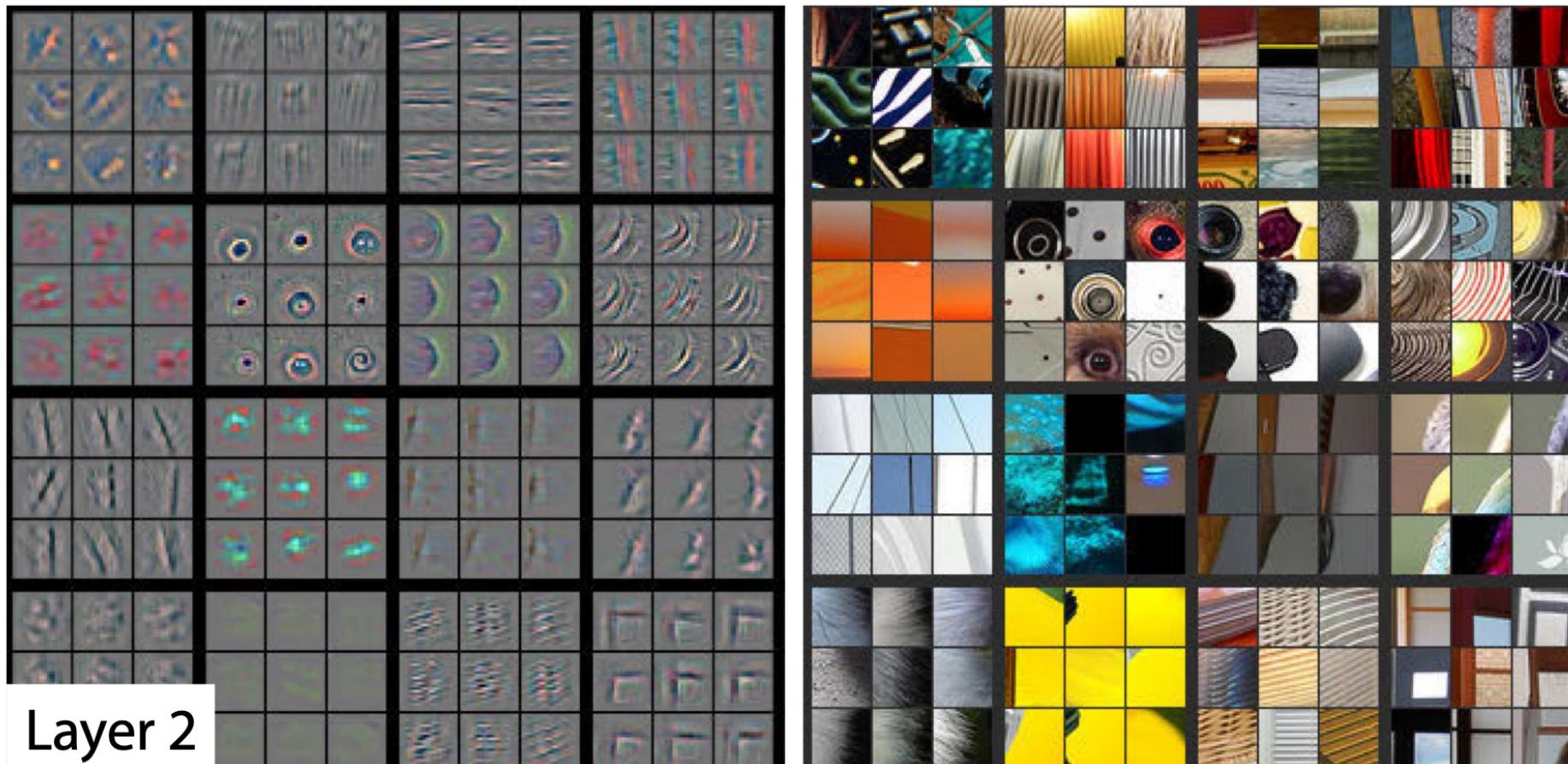
# Представления с последних слоёв



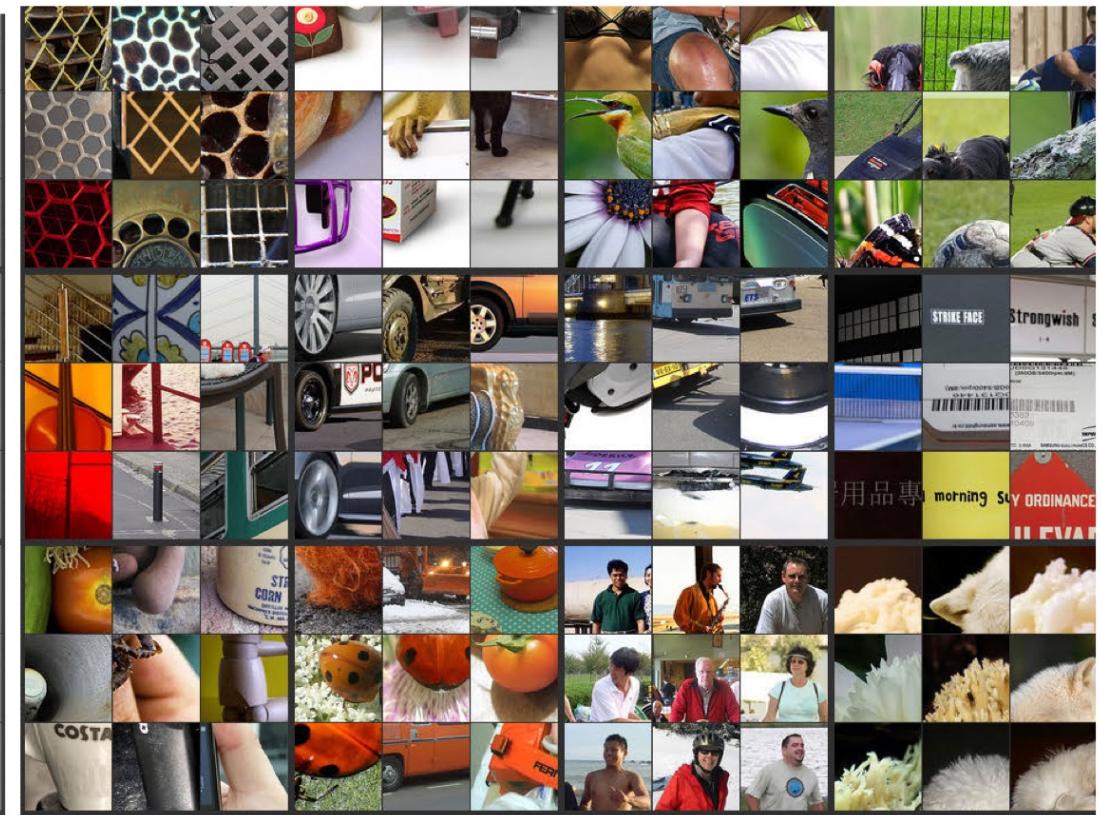
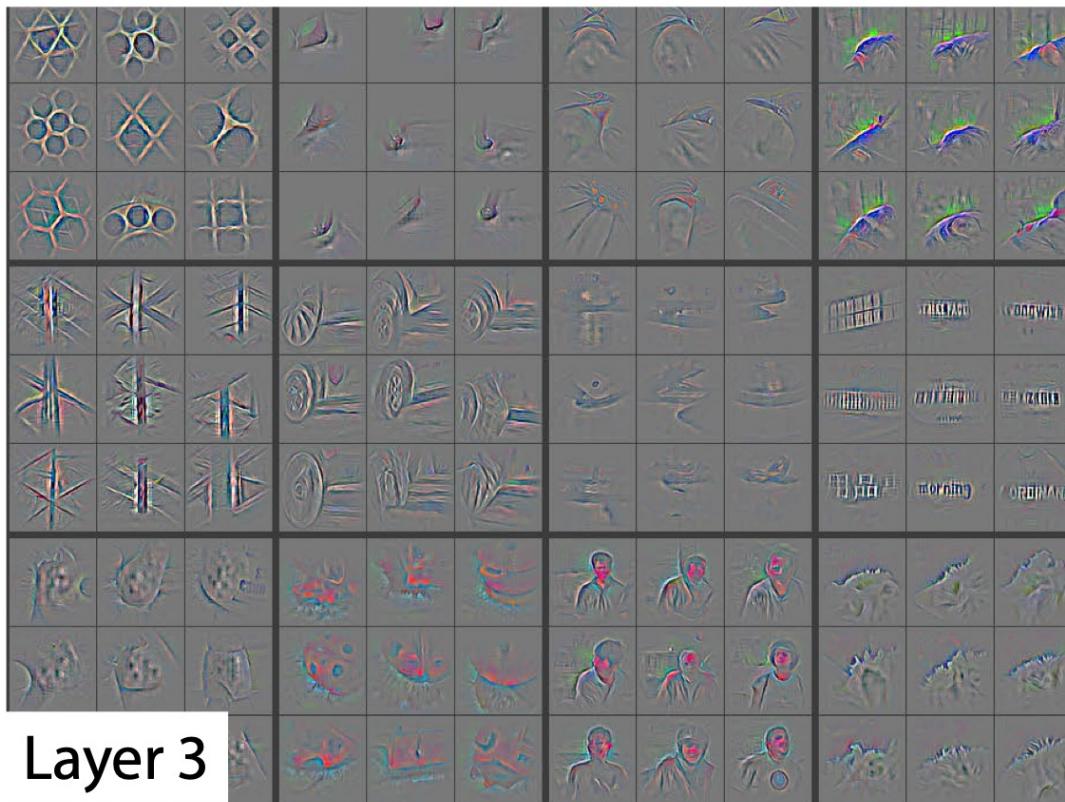
Layer 1



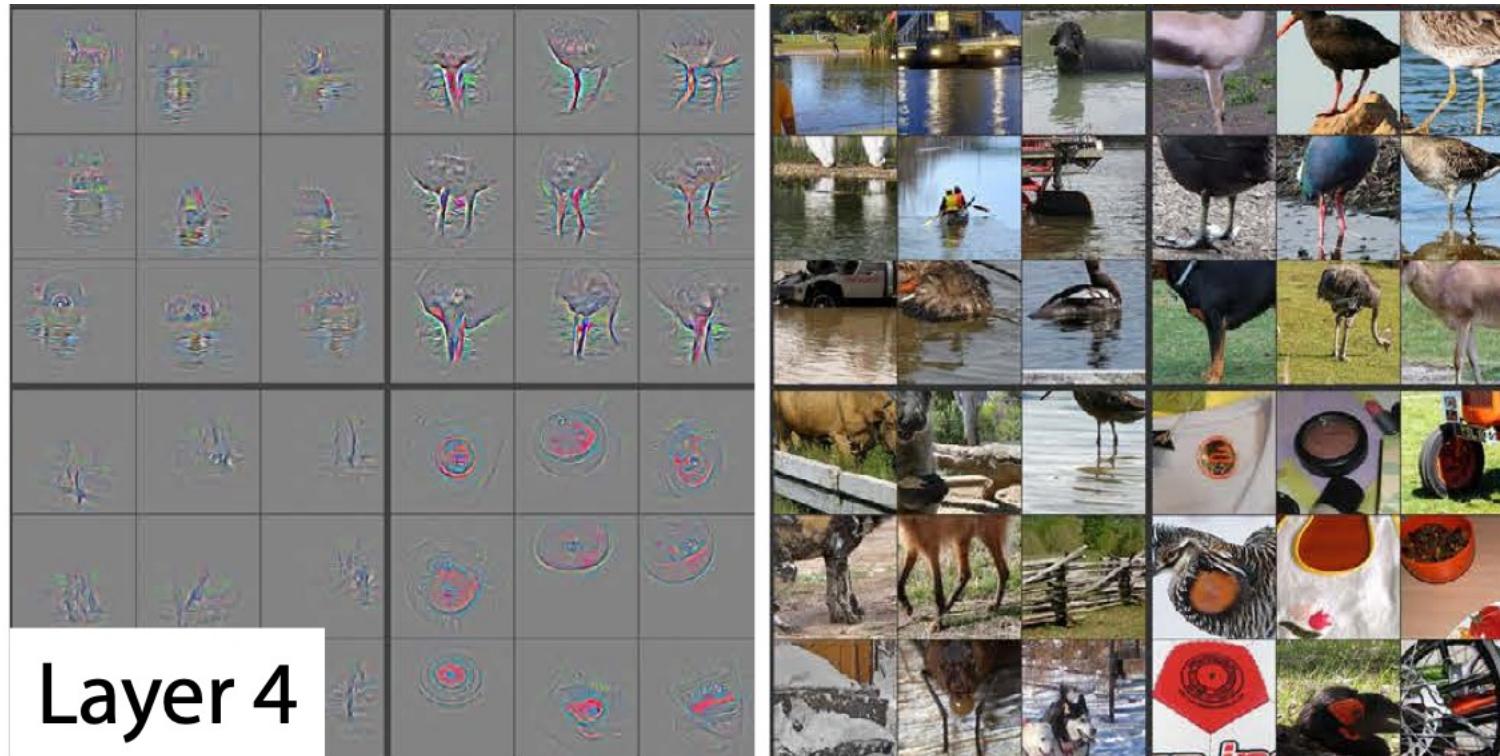
# Представления с последних слоёв



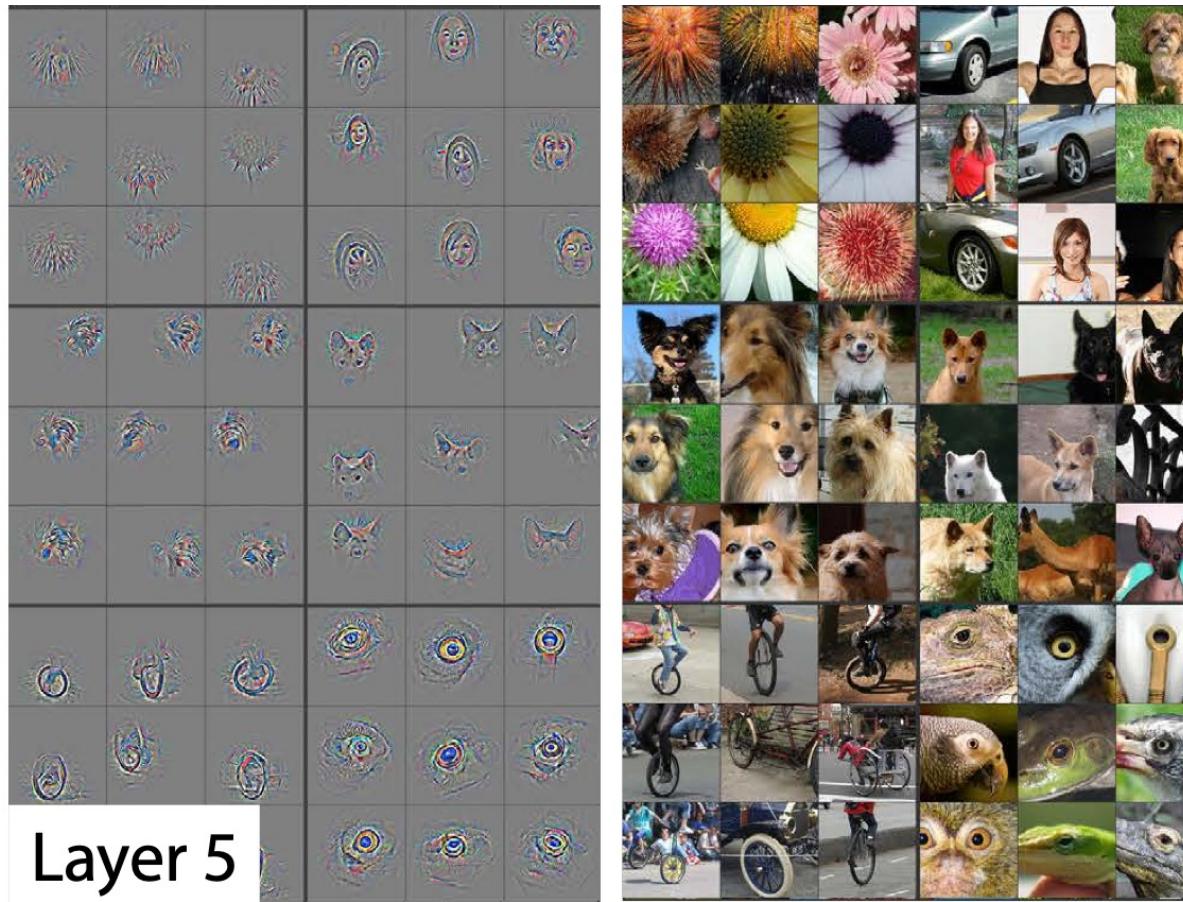
# Представления с последних слоёв



# Представления с последних слоёв



# Представления с последних слоёв



# Стохастический градиентный спуск

# Градиентный спуск

1. Начальное приближение:  $w^0$

2. Повторять:

$$w^t = w^{t-1} - \eta \nabla Q(w^{t-1})$$

3. Останавливаемся, если ошибка на тестовой выборке перестаёт убывать

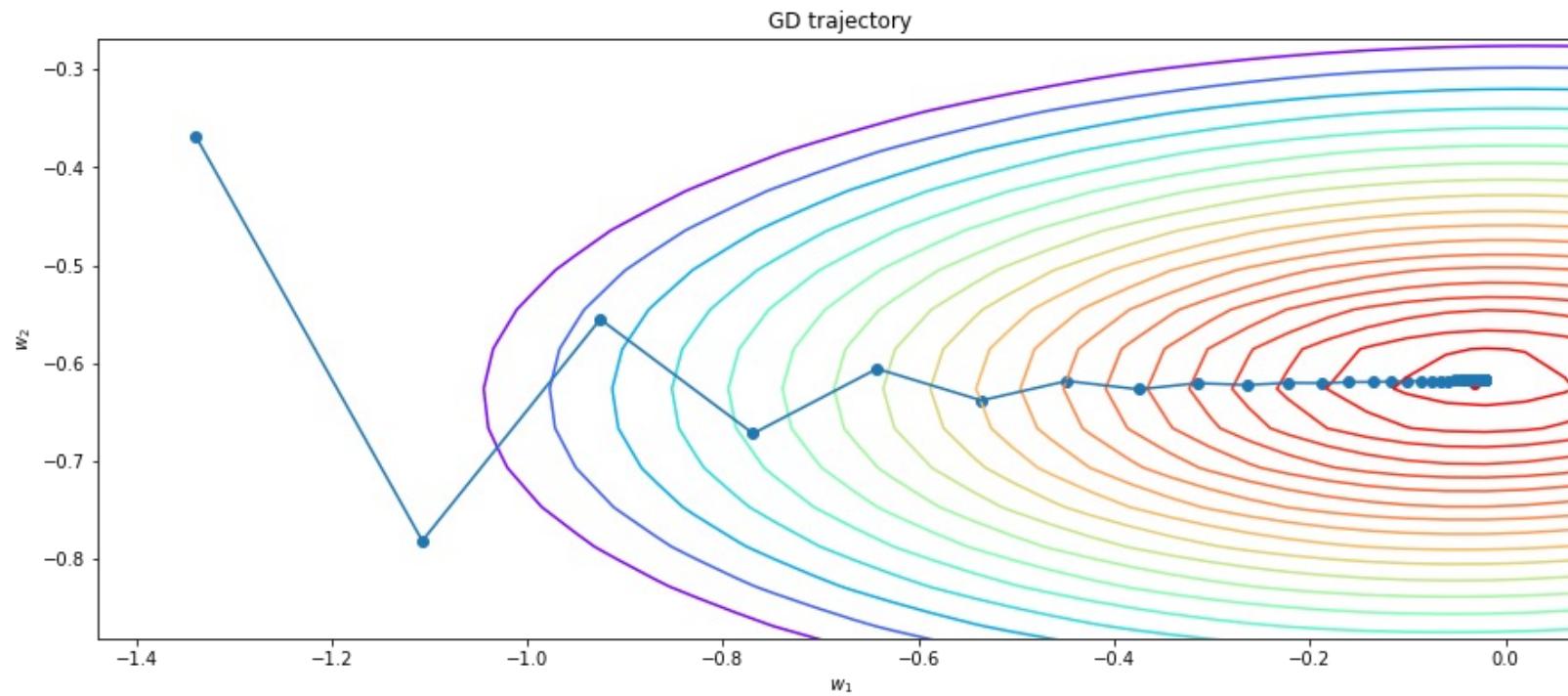
# Стохастический градиентный спуск

1. Начальное приближение:  $w^0$
2. Повторять, каждый раз выбирая случайный объект  $i_t$ :

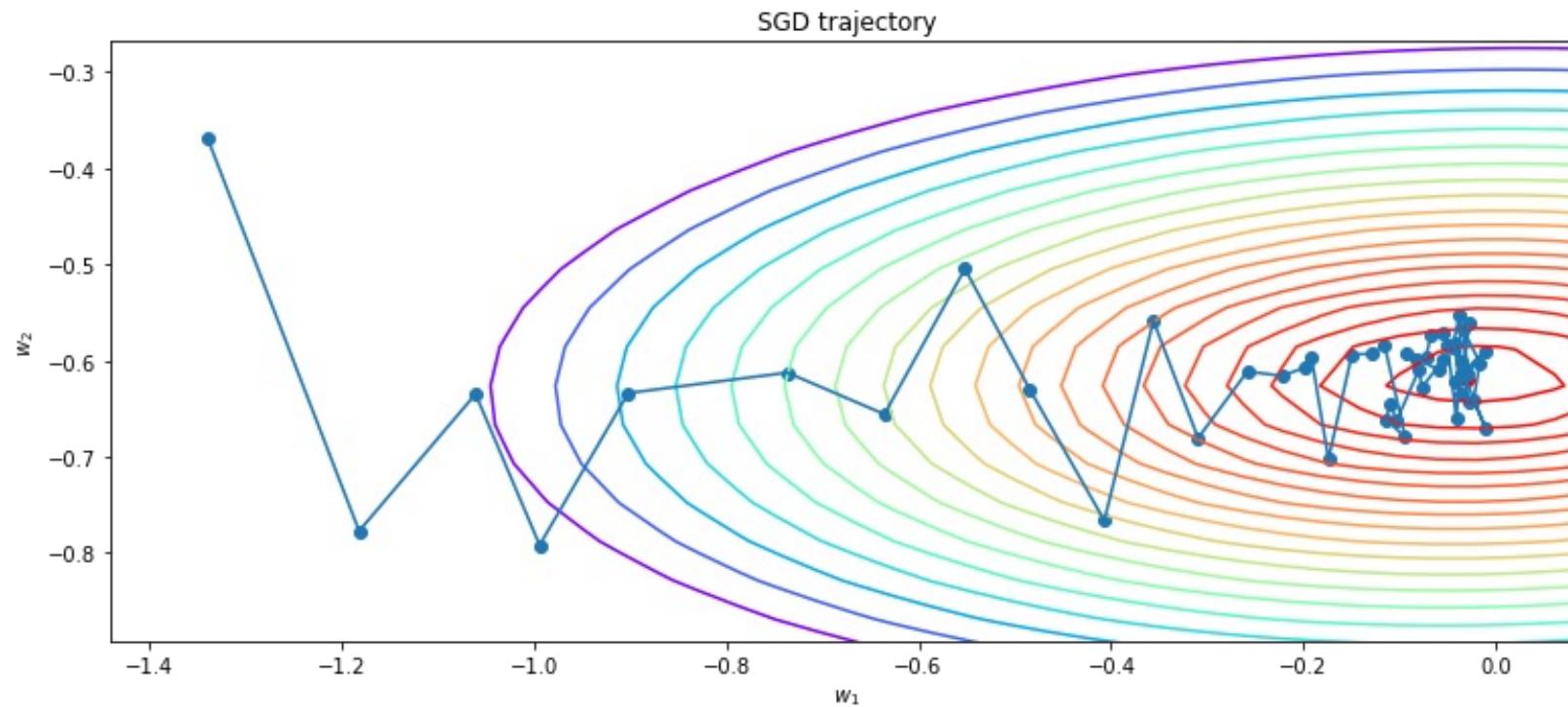
$$w^t = w^{t-1} - \eta \nabla L(y_{i_t}, a(x_{i_t}))$$

3. Останавливаемся, если ошибка на тестовой выборке перестаёт убывать

# Градиентный спуск



# Стochastic gradient descent



# Стохастический градиентный спуск

1. Начальное приближение:  $w^0$
2. Повторять, каждый раз выбирая случайный объект  $i_t$ :

$$w^t = w^{t-1} - \eta_t \nabla L(y_{i_t}, a(x_{i_t}))$$

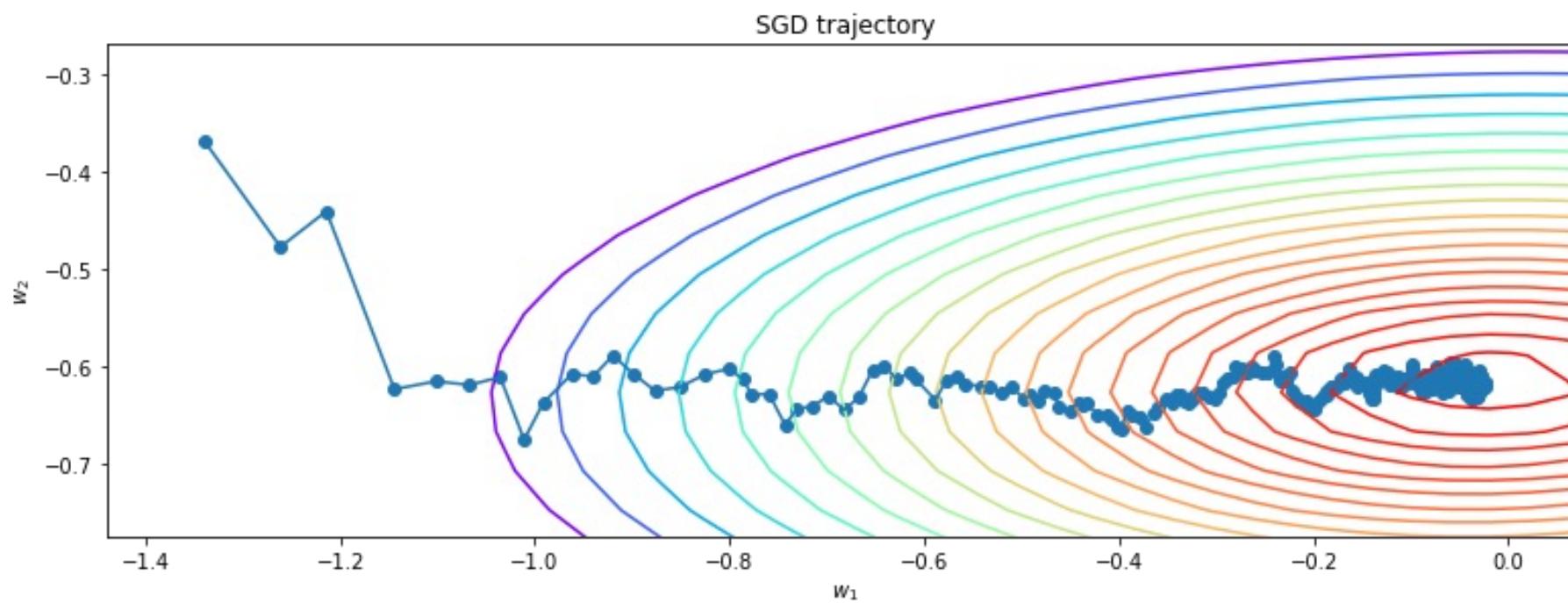
3. Останавливаемся, если ошибка на тестовой выборке перестаёт убывать

# Стохастический градиентный спуск

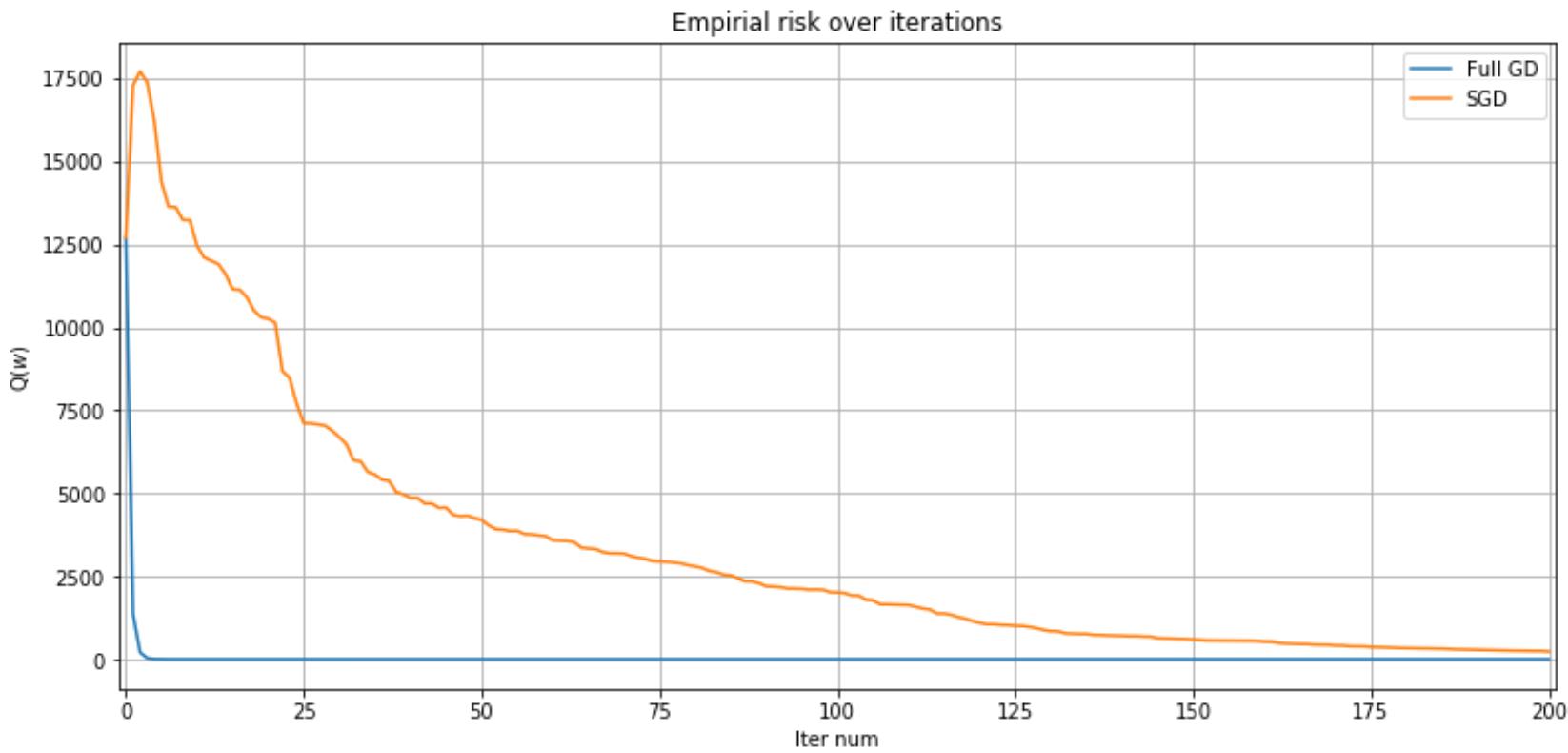
- Оценка по одному объекту **несмещённая**
- То есть в среднем мы идём в правильную сторону
- Даже в точке оптимума оценка по одному объекту вряд ли будет нулевой
- Поэтому важно, чтобы длина шага стремилась к нулю
- Сходимость к глобальному минимуму гарантируется только для выпуклых функций

# Стochastic gradient descent

$$\eta_t = \frac{0.1}{t^{0.3}}$$



# Стochastic gradient descent



# Mini-batch GD

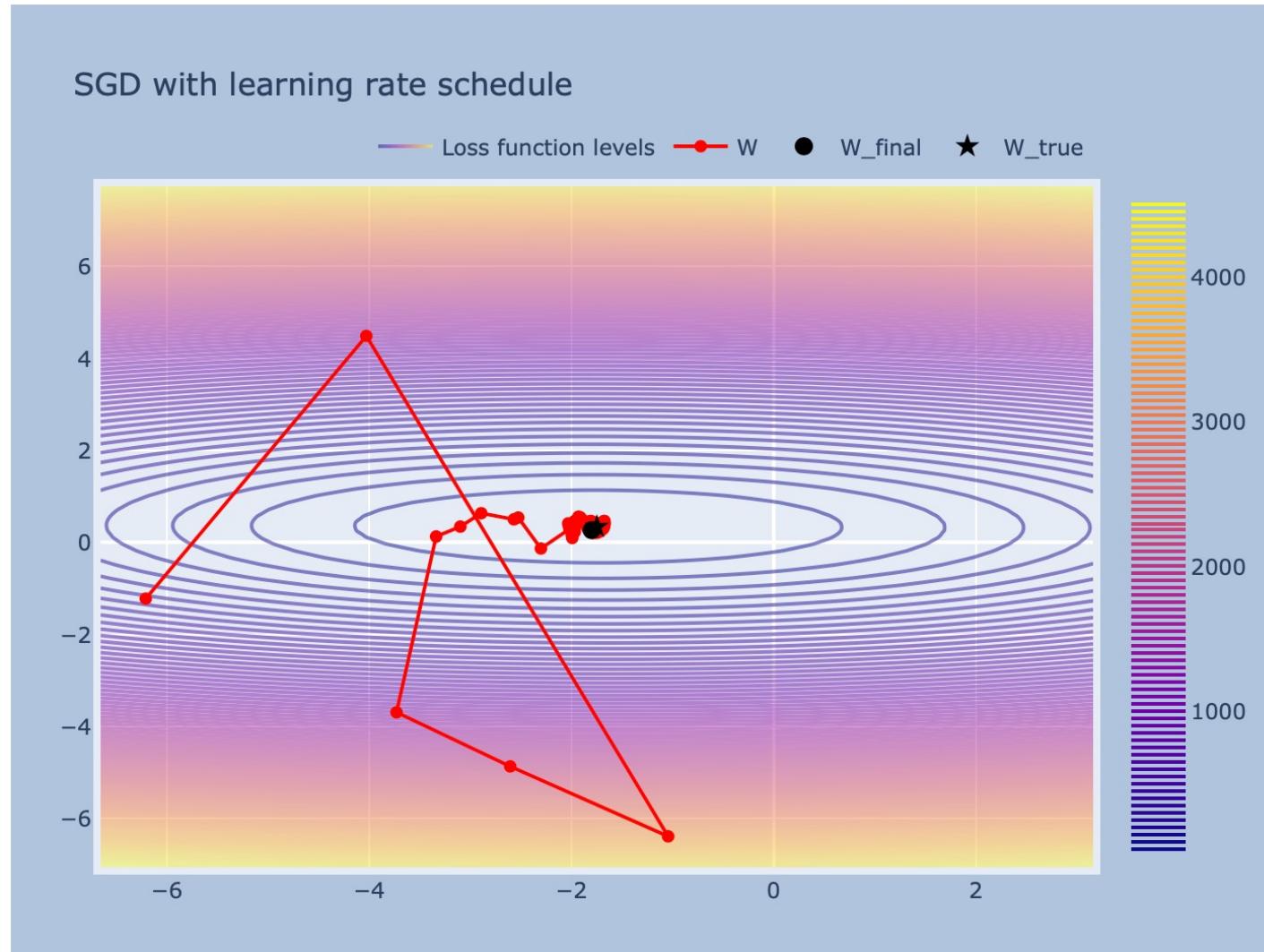
1. Начальное приближение:  $w^0$
2. Повторять, каждый раз выбирая  $t$  случайных объектов  $i_1, \dots, i_m$ :

$$w^t = w^{t-1} - \eta_t \frac{1}{n} \sum_{j=1}^n \nabla L(y_{t,j}, a(x_{t,j}))$$

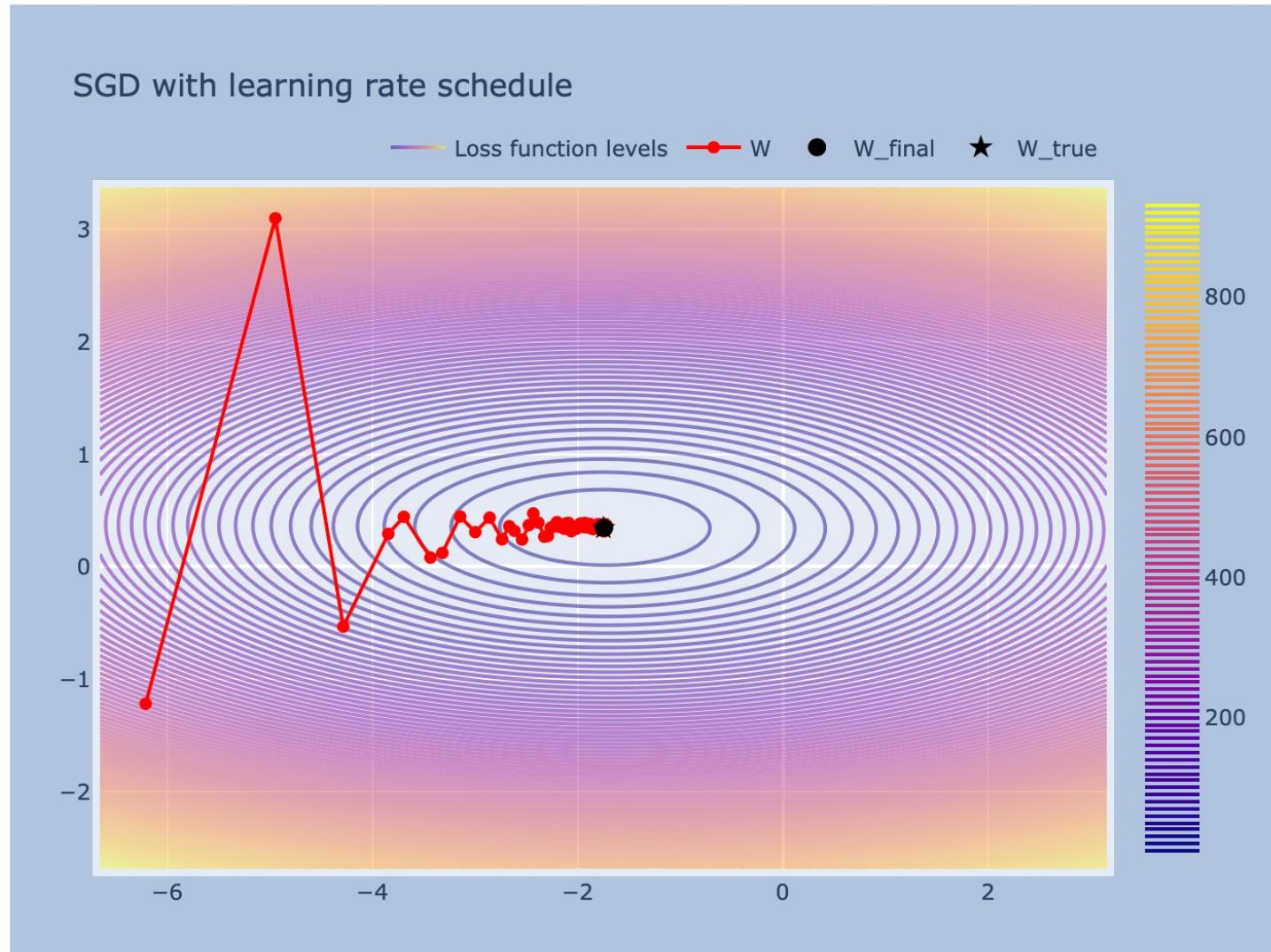
3. Останавливаемся, если ошибка на тестовой выборке перестаёт убывать

$x_{t,j}$  — объект номер  $j$  из батча, сформированного на шаге  $t$

# Батч размера 1



# Батч размера 10



# Batch size

- Размер пакета — обычно порядка десятков или сотни
- Имеет смысл брать степень двойки
- Возможно, делает оценку градиента более стабильной
- Вычислительно почти так же эффективен, как шаг по градиенту одного объекта — за счёт векторизации

# Batch size

The collected experimental results for the CIFAR-10, CIFAR-100 and ImageNet datasets show that increasing the mini-batch size progressively reduces the range of learning rates that provide stable convergence and acceptable test performance. On the other hand, small mini-batch sizes provide more up-to-date gradient calculations, which yields more stable and reliable training. The best performance has been consistently obtained for mini-batch sizes between  $m = 2$  and  $m = 32$ , which contrasts with recent work advocating the use of mini-batch sizes in the thousands.

# Batch size



Yann LeCun

April 27, 2018 ·

...

Training with large minibatches is bad for your health.

More importantly, it's bad for your test error.

Friends dont let friends use minibatches larger than 32.

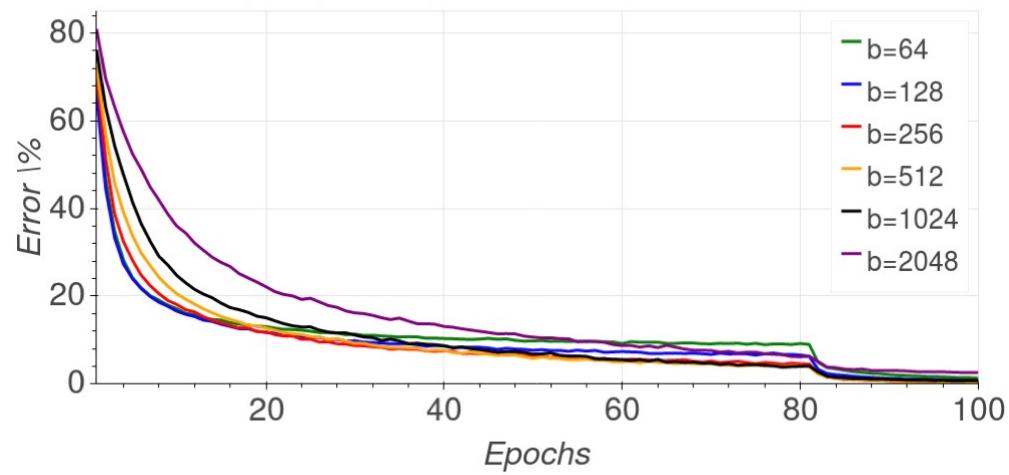
Let's face it: the \*only\* people have switched to minibatch sizes larger than one since 2012 is because GPUs are inefficient for batch sizes smaller than 32. That's a terrible reason. It just means our hardware sucks.

What's worse is that the easiest way to parallelize training is to make the minibatch even larger and distribute it across multiple GPUs and multiple nodes.

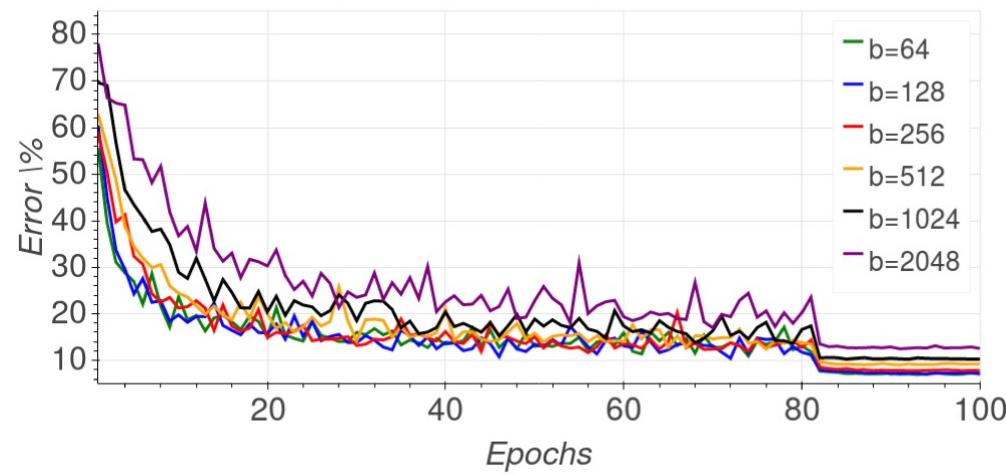
Minibatch sizes over 1024 aren't just bad for your health. They cause brain tumors.

They learn quickly, but the wrong thing.

# Batch size

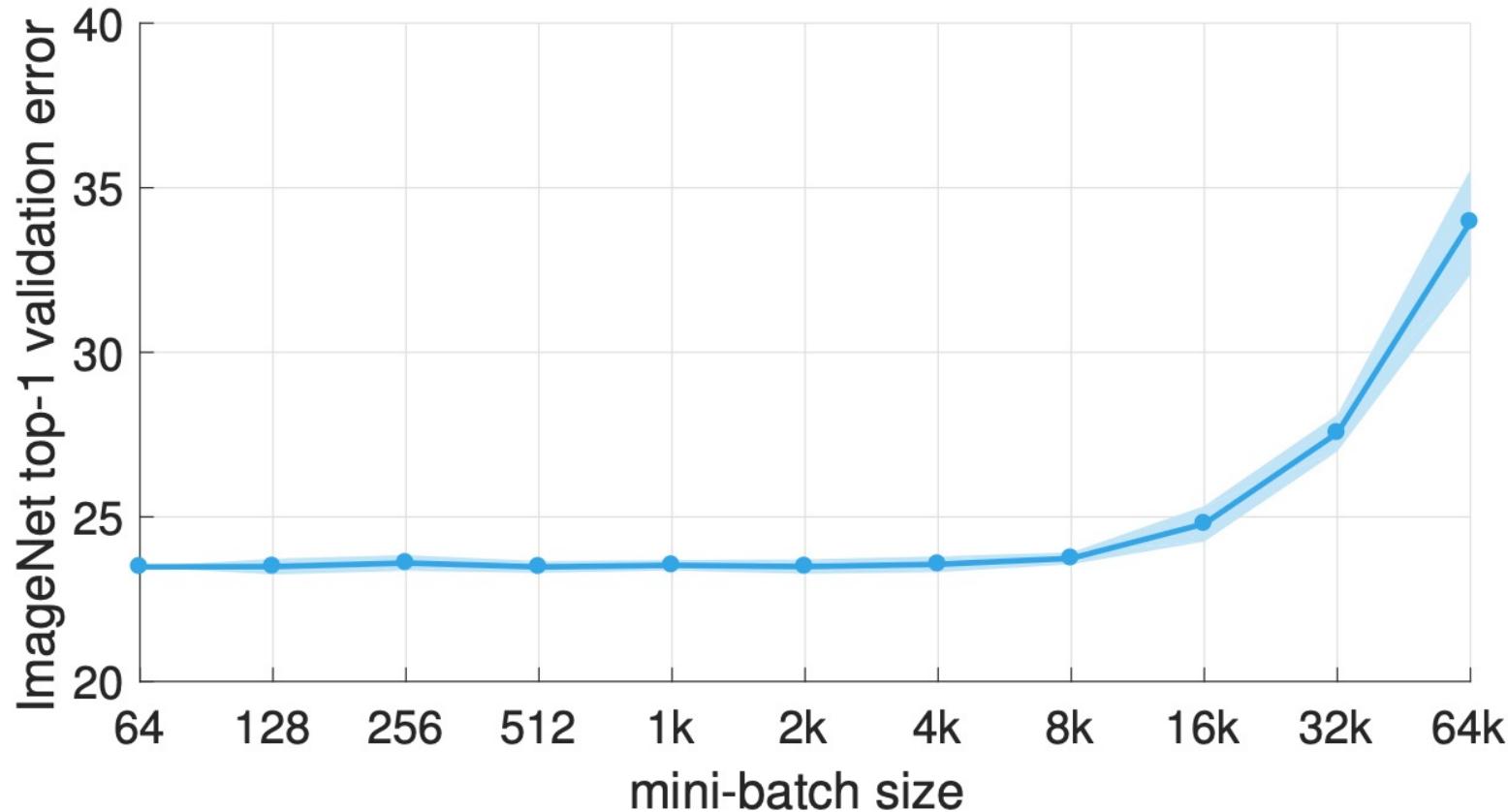


(a) Training error

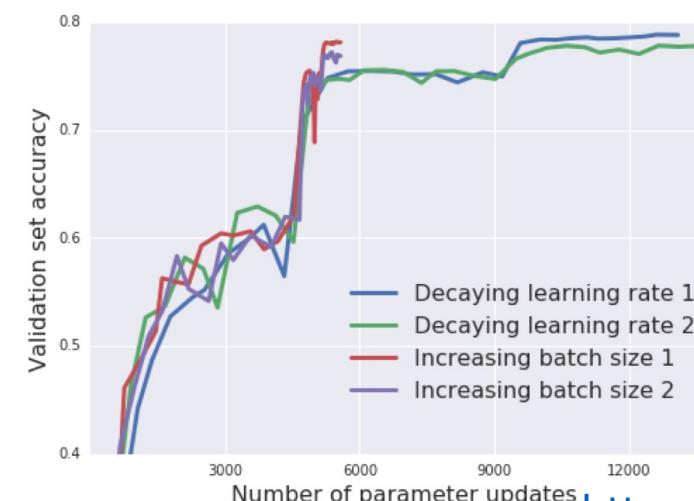
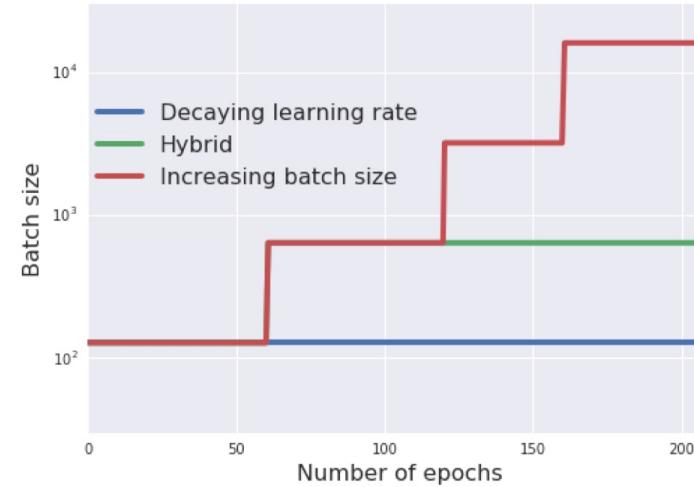
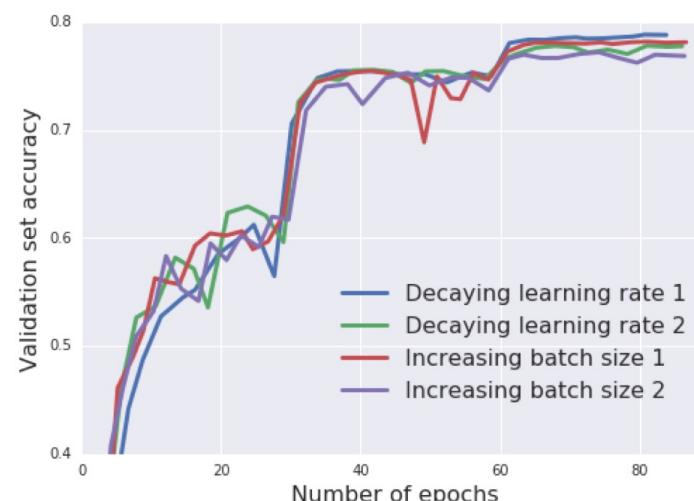
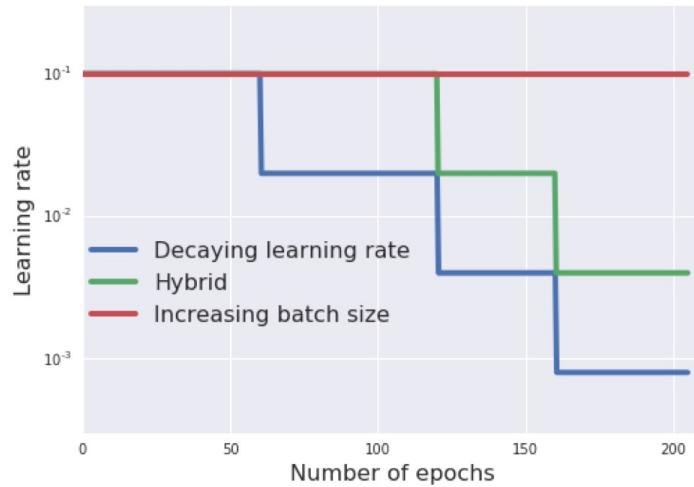


(b) Validation error

Надо грамотно подбирать формулу для длины шага!



# Batch size

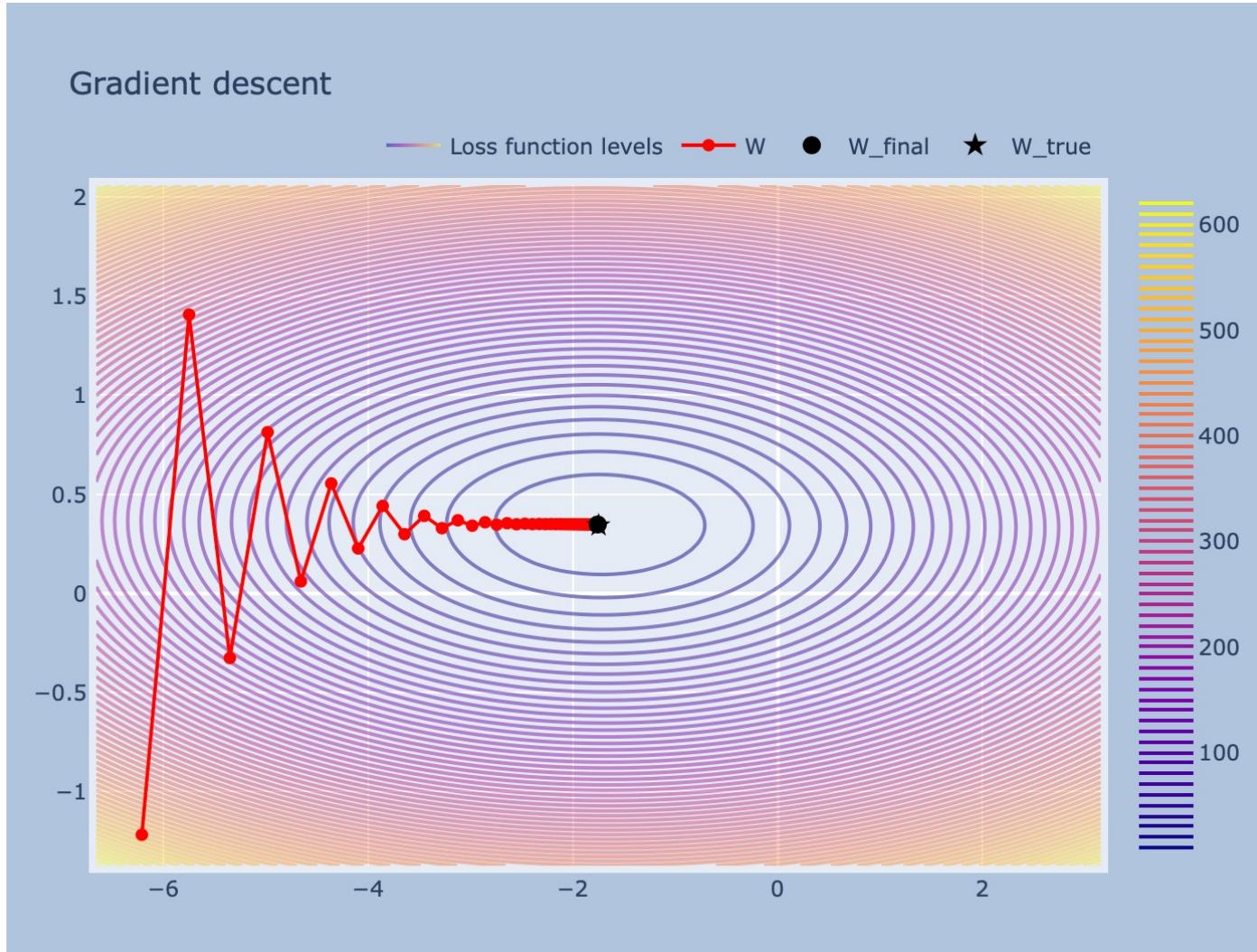


# Batch size

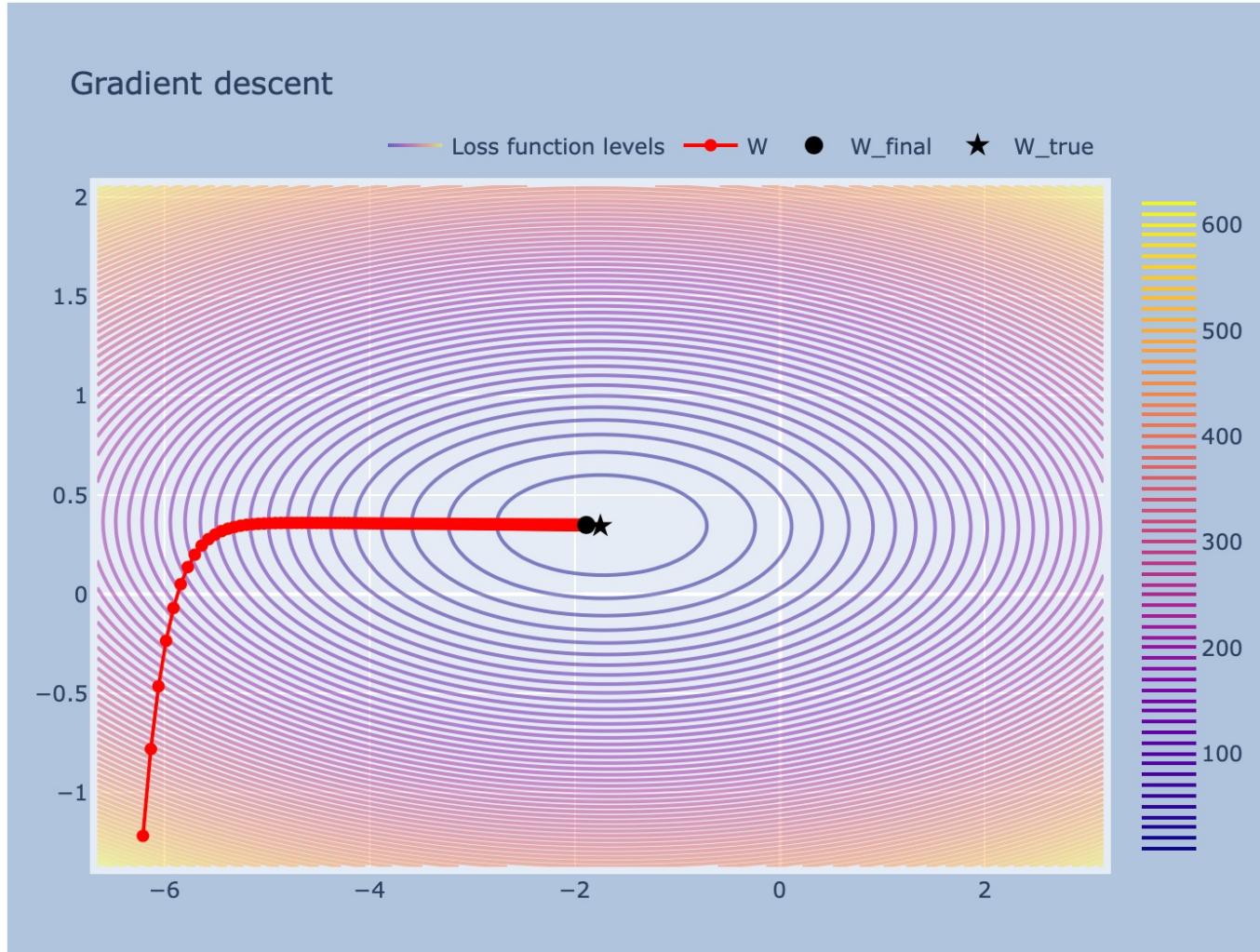
- От большого размера батча может быть польза
- Для этого надо грамотно подбирать длину шага и/или постепенно увеличить размер батча

# Модификации градиентного спуска

# Проблемы



# Проблемы



# Проблемы

- Если у функции «вытянуты» линии уровня, то градиентный спуск требует аккуратного выбора длины шага и будет долго сходиться

# Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$

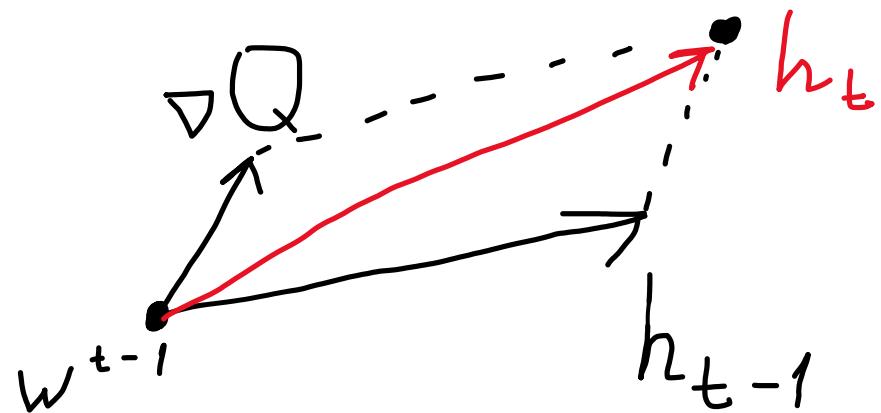
$$w^t = w^{t-1} - h_t$$

- $h_t$  — «инерция», усреднённое направление движения
- $\alpha$  — параметр затухания
- Как будто шарик, который катится в сторону минимума, очень тяжёлый

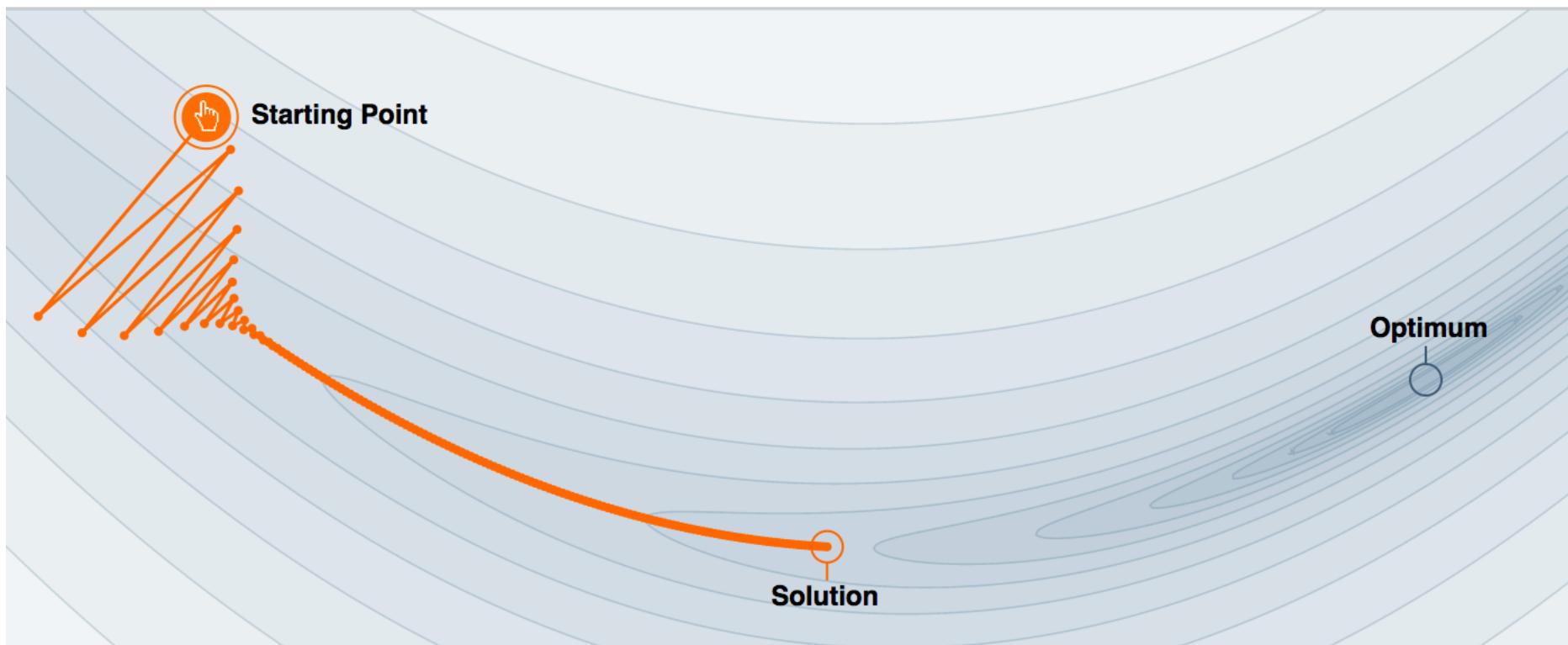
# Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$

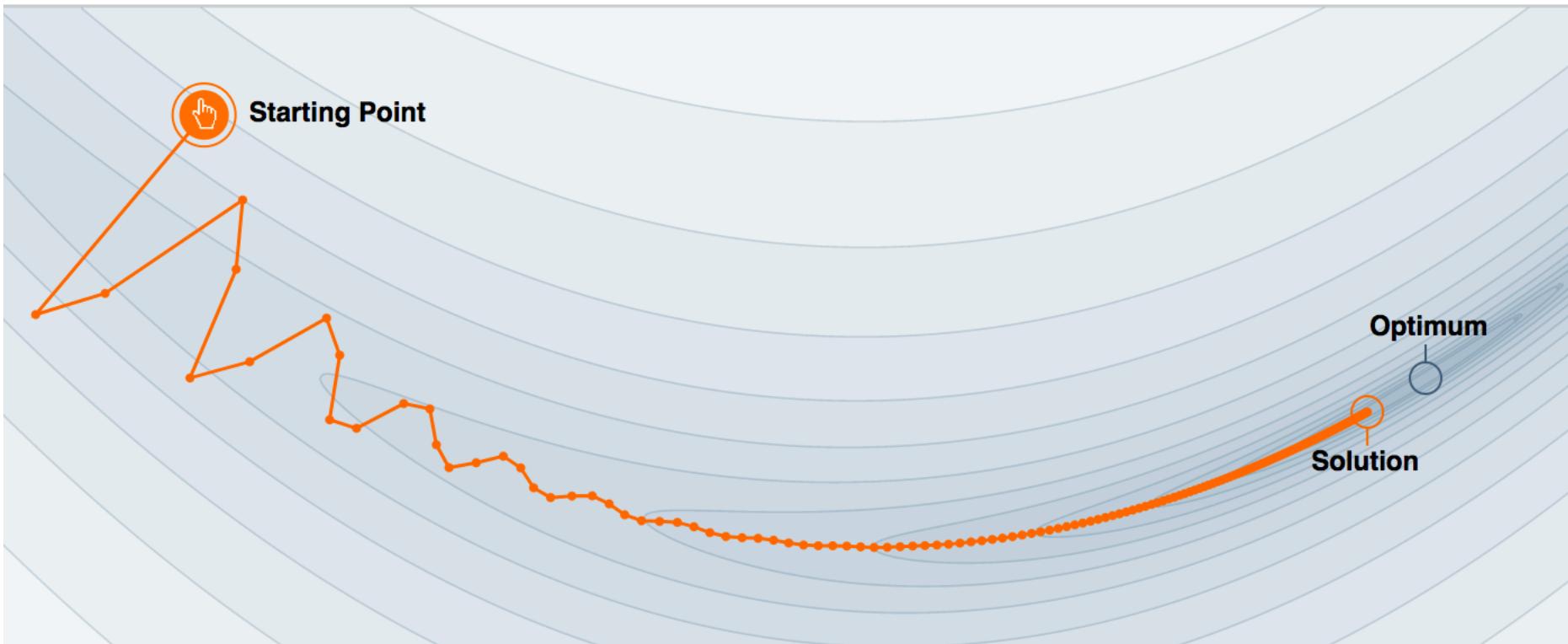
$$w^t = w^{t-1} - h_t$$



# Без инерции



# Синерцией



# Nesterov Momentum

$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1} - \alpha h_{t-1})$$

$$w^t = w^{t-1} - h_t$$

- $w^{t-1} - \alpha h_{t-1}$  — неплохая оценка того, куда мы попадём на следующем шаге

